



BLUEPIRAT Client Library 5.0.4

User's manual

Generated by Doxygen 1.8.0

Mon Dec 7 2020 12:21:33

Contents

1	User's manual - BLUEPIRAT Client Library 5.0.4	1
1.1	General	1
1.2	Functionality	1
1.3	Compiler/Linker	2
1.4	Thread safety	2
1.5	Demo project	2
2	Deprecated List	17
3	Hierarchical Index	19
3.1	Class Hierarchy	19
4	Class Index	21
4.1	Class List	21
5	File Index	23
5.1	File List	23
6	Class Documentation	25
6.1	BPNGError Struct Reference	25
6.2	BPNGLoggerDetector Class Reference	25
6.3	DataSpan Struct Reference	32
6.4	IBPNGClient Struct Reference	32
6.5	IBPNGClientListener Struct Reference	50
6.6	IChannel Struct Reference	56
6.7	IChannelList Struct Reference	56
6.8	IClientProperties Struct Reference	57
6.9	IConversionSet Struct Reference	64
6.10	IFalseMeasureSignal Struct Reference	66
6.11	IFalseMeasureSignalList Struct Reference	66
6.12	IFormatInfo Struct Reference	66
6.13	IFormatList Struct Reference	67
6.14	IRdbEvent Struct Reference	67
6.15	IRdbEventList Struct Reference	68
6.16	IRdbTraceBlock Struct Reference	69
6.17	IRdbTraceBlockList Struct Reference	69
6.18	ITesttoolsChannel Struct Reference	69
6.19	ITesttoolsChannelList Struct Reference	70
6.20	LogInData Struct Reference	71
6.21	MemoryFillLevel Struct Reference	71
6.22	OnlineLoggerInfo Struct Reference	72
6.23	OnlineLoggerInfoStringPair Struct Reference	73
6.24	RdbEvent2 Struct Reference	74

6.25 RdbEventList Class Reference	75
6.26 TSLCluster Struct Reference	75
7 File Documentation	77
7.1 BPNGDefines.h File Reference	77
7.2 BPNGLoggerDetector.hh File Reference	86
7.3 IBPNGClient.h File Reference	86
7.4 IBPNGClientListener.h File Reference	89
7.5 IClientProperties.h File Reference	89
7.6 RdbDefines.h File Reference	90
7.7 RdbEventList.hh File Reference	91
Index	93

Chapter 1

User's manual - BLUEPIRAT Client Library 5.0.4

1.1 General

This is the documentation for the C++ BLUEPIRAT 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. [IConversionSet](#), 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 BLUEPIRAT Client Library provides methods for base functionality like:

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

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

Exampe for lib unsage:

```
//*****
//
// main.cc
//
//*****

// sys
#include <sys/stat.h>
#include <cerrno>
#include <ctime>
#include <cstring>
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>

// tmlib
#ifdef _MSC_VER
#include <fileutils.hh>
#endif

// atom
#include "BPNGDefines.h"

// client
#include "windirent.h"
#include "IBPNGClientListener.h"
#include <IBPNGClient.h>
#include "BPNGLoggerDetector.hh"
#include "RdbEventList.hh"
#include "TSLClusterImpl.hh"
#include "miniz.h"

#ifdef _MSC_VER
#include <sys/stat.h>
#else
#include <direct.h>
#define mkdir(a, b) _mkdir(a)
#endif

using namespace std;

/*****ONLINE*DOWNLOAD*****/
static void sampleFunctionDownload(OnlineLoggerInfo device);
static void sampleFunctionTSLDownload(TSLClusterImpl tsl);
/*****ONLINE*CONVERSION*****/
```

```

static void sampleFunctionOnlineConversion(OnlineLoggerInfo device);
static void sampleFunctionTSLOnlineConversion(TSLClusterImpl tsl);
/*****CONFIG*****/
static void sampleFunctionConfiguration(OnlineLoggerInfo device);
static void sampleFunctionTSLConfiguration(TSLClusterImpl tsl);
/*****OFFLINE*CONVERSION*****/
/*****/
static void sampleFunctionOfflineConversion();
static void sampleFunctionOfflineTSLConversion();
/*****HELPERS*****/
/*****/
static string getLocalDateString();
static vector<string> readZipsFromDirectory(string dir);
static bool extractZipToDirectory(string zipPath, string targetPath);

int main()
{
    // Get list of all currently available blue PiraT 2 devices
    BPNGLoggerDetector detector;
    vector<OnlineLoggerInfo> devices = detector.getLoggerList(0);
    vector<TSLClusterImpl> tsls = detector.getTSLs(devices);

    // select the device you want to work with
    OnlineLoggerInfo device;
    TSLClusterImpl targetTsl;
    bool found = false;
    bool tslFound = false;

    //map for tsl's
    for (size_t i = 0; i < devices.size(); ++i)
    {
        if (strcmp(devices[i].ip, "192.168.0.233") == 0)
        {
            device = devices[i];
            found = true;
            break;
        }
    }

    for (size_t i = 0; i < tsls.size(); ++i)
    {
        TSLClusterImpl tsl = tsls[i];
        cout << "Found TSL [" << tsl.getTSLName() << "] with devices\n";

        tsl.print();

        cout << "\n";

        /* insert target tsl name here*/
        if (tsl.getTSLName() == "MyTSL")
        {
            targetTsl = tsl;
            tslFound = true;
        }
    }

    /* activate one of the sample functions */
    if (found)
    {
        //sampleFunctionDownload(device);
        //sampleFunctionOnlineConversion(device);
        //sampleFunctionConfiguration(device);
    }

    if (tslFound)
    {
        //sampleFunctionTSLConfiguration(targetTsl);
        //sampleFunctionTSLDownload(targetTsl);
        //sampleFunctionTSLOnlineConversion(targetTsl);
    }

    /* offline samples*/
    //sampleFunctionOfflineConversion();
    //sampleFunctionOfflineTSLConversion();

    int ret = system("pause");
    if (ret == -1)
        cerr << "error in system(\"pause\"): " << strerror(errno) << endl;
}

```

```

}

/*****ONLINE*DOWNLOAD*****/
*****/

// We want to download all traces since last startup to an offline data set
#ifdef _MSC_VER
static void sampleFunctionDownload(OnlineLoggerInfo device) __attribute__((unused));
#endif

static void sampleFunctionDownload(OnlineLoggerInfo device)
{
    IBPNGClient *client = getBPNGClient();

    // connect logger
    BOOL ret = client->connectLogger(1, &device);
    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->initialize();
    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 == 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", 0x777);
    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 << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

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

// We want to download all traces since last startup to an offline data set
#ifdef _MSC_VER
static void sampleFunctionTSLDownload(TSLClusterImpl tsl) __attribute__((unused));
#endif

static void sampleFunctionTSLDownload(TSLClusterImpl tsl)
{
    IBPNGClient *client = getBPNGClient();
    client->setTSLCluster(tsl.getTSLCluster());

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

    ret = client->initialize();
    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 == 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", 0x777);
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\\
TSL_Offline.zip", 0);
if (ret == 0)
{
    BPNGError err = client->getLastError();
    cout << "Failed to download data." << endl;
    cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
    client->disconnectLogger();
    client->release();
    return;
}

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

/*****
*****ONLINE*CONVERSION*****
*****/

// We want to convert all CAN traces from the logger
// around the last Marker to CANoe asc and BLF format.
#ifdef _MSC_VER
static void sampleFunctionOnlineConversion(OnlineLoggerInfo device) __attribute__((unused))
;
#endif

static void sampleFunctionOnlineConversion(OnlineLoggerInfo device)
{
    IBPNGClient *client = getBPNGClient();

    // connect logger
    BOOL ret = client->connectLogger(1, &device);
    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->initialize();
    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 markerTimeStamp = 0;
    // search last marker
    for (int i = eventList.size() - 1; i >= 0; --i)
    {
        if (eventList[i].type == 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", 0x777);
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_CAN)
        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,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    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
        // fileId to the same output file (if procurable in accordance with
        // the format specification.
        conversionSet->addChannel(type,
            index,
            CANOE,
            10,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    else
    {
        // All other CAN channels to one BLF file.
        conversionSet->addChannel(type,

```

```

        index,
        BLF,
        20,
        channels->getChannel(i)->getOffset(),
        channels->getChannel(i)->getMainboardNumber(),
        channels->getChannel(i)->isMappingActive(),
        channels->getChannel(i)->getMappedChannelIndex());
    }
}

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();
}

#ifdef _MSC_VER
static void sampleFunctionTSLOnlineConversion(TSLClusterImpl tsl) __attribute__((unused));
#endif

static void sampleFunctionTSLOnlineConversion(TSLClusterImpl tsl)
{
    IBPNGClient *client = getBPNGClient();
    client->setTSLCluster(tsl.getTSLCluster());

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

    ret = client->initialize();
    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 markerTimeStamp = 0;
    // search last marker
    for (int i = eventList.size() - 1; i >= 0; --i)
    {
        if (eventList[i].type == 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", 0x777);
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.
// On TSL we have to add offset and mainboardnumber for channel identification.
const IChannelList* channels = client->getLoggerChannels();
for (int i = 0; i < channels->getSize(); ++i)
{
    ChannelType type = channels->getChannel(i)->getType();
    if (type != CH_CAN)
        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,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    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,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    else
    {
        // All other CAN channels to one BLF file.
        conversionSet->addChannel(type,
            index,
            BLF,
            20,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
}

```

```

        channels->getChannel(i)->getMappedChannelIndex());
    }
}

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();
}

/***** CONFIG *****/

// This function shows how to:
// - download the configuration
// - reconfigure the logger device
// - set the default config
#ifdef _MSC_VER
static void sampleFunctionConfiguration(OnlineLoggerInfo device) __attribute__((unused));
#endif

static void sampleFunctionConfiguration(OnlineLoggerInfo device)
{
    IBPNGClient *client = getBPNGClient();
    client->setDevice(device);
    BOOL ret = client->connect();
    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
    stringstream targetPath;
    targetPath << "..\\testoutdir\\bpng_[" << device.mbnr << "][" << getLocalDateString() << "].zip";
    ret = client->getConfig(targetPath.str().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().
    // the new config archive needs a date in its filename in this form: YYYY-MM-DD_HH-MM-SS

    // We use the same config that we downloaded
    string newConfigPath = targetPath.str();
    OnlineLoggerInfoStringPair pair;
    pair.key = device;
    pair.value = newConfigPath.c_str();
    ret = client->reconfigLogger(1, &pair);
    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 << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

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

// This function shows how to:
// - download the configuration
// - reconfigure the logger device
// - set the default config
#ifdef _MSC_VER
static void sampleFunctionTSLConfiguration(TSLClusterImpl tsl) __attribute__((unused));
#endif

static void sampleFunctionTSLConfiguration(TSLClusterImpl tsl)
{
    //get the tsl client instance
    IBPNGClient *client = getBPNGClient();
    client->setTSLCluster(tsl.getTSLCluster());

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

    // save current config
    string targetPath = "..\\testoutdir\\tsl_[" + tsl.getTSLName() + "][" + getLocalDateString() + "];";
    string zipPath = targetPath + ".zip";
    ret = client->getConfig(zipPath.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to download configurations." << 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().
    // the new config archive needs a date in its filename in this form: YYYY-MM-DD_HH-MM-SS

    // We use the same config that we downloaded
    // for tsl we have to extract the ZIP first.
    targetPath += "\\ ";
    extractZipToDirectory(zipPath, targetPath);
    //create one string which includes all config-ip pairs
    vector<string> configZips = readZipsFromDirectory(targetPath);

    // For reasons of simplicity we use the devices' mainboard number to determine
    // Which configuration part (ZIP) belongs to which device. If you want to send
    // the same configuration to different TSL clusters this will not work.
    vector<string> paths;
    vector<OnlineLoggerInfoStringPair> pairs;

    for (vector<OnlineLoggerInfo>::iterator iter = tsl.begin(); iter != tsl.end(); ++iter)
    {

```

```

OnlineLoggerInfo device = *iter;
for (size_t i = 0; i != configZips.size(); ++i)
{
    string configZip = configZips[i];
    if (configZip.find(device.mbnr) != string::npos)
    {
        OnlineLoggerInfoStringPair pair;
        pair.key = device;
        paths.push_back(targetPath + configZip);
        pair.value = paths.back().c_str();
        pairs.push_back(pair);
        break;
    }
}

ret = client->reconfigLogger(pairs.size(), &pairs[0]);
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;
}

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

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

/*****OFFLINE*CONVERSION*****/

// We want to convert all CAN traces from an Offline data set
// around the last Marker to CANoe asc and BLF format.
#ifdef _MSC_VER
static void sampleFunctionOfflineConversion() __attribute__((unused));
#endif

static 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.
    client->setOfflineData("../testoutdir\\BP2_Offline.zip");
    BOOL ret = client->initialize();
    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 == 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", 0x777);
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_CAN)
        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,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    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,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    else
    {

```

```

        // All other CAN channels to one BLF file.
        conversionSet->addChannel(type,
            index,
            BLF,
            20,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
}

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();
}

// We want to convert all A/I, D/I traces from an Offline TSL data set
// to XTMT
#ifdef _MSC_VER
static void sampleFunctionOfflineTSLConversion() __attribute__((unused));
#endif

static void sampleFunctionOfflineTSLConversion()
{
    // We use the sample Offline Data Set that was downloaded
    // with sampleFunctionTSLDownload().
    string offlineDataSetPath = "..\\testoutdir\\TSL_Offline.zip";

    IBPNGClient *client = getBPNGClient();
    // Its up to you to ensure an existing file.
    client->setOfflineData(offlineDataSetPath.c_str());
    BOOL ret = client->initialize();
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to init offline." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

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

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

    //select all spans
    conversionSet->addTimeSpan(0, 0xFFFFFFFFFFFFFFFFUL);

    // All A/I, D/I channels are supposed to be written in separated XTMT files.
    const IChannelList* channels = client->getLoggerChannels();
    for (int i = 0; i < channels->getSize(); ++i)
    {
        ChannelType type = channels->getChannel(i)->getType();
        if (type == CH_DIGITAL_IN || type == CH_ANALOG_IN)
        {
            // -1 as fileId parameter creates a separate file for this channel
            // for tsl the offset and mainboardnumber fields are needed
            conversionSet->addChannel(type,

```

```

        channels->getChannel(i)->getIndex(),
        XTMT,
        -1,
        channels->getChannel(i)->getOffset(),
        channels->getChannel(i)->getMainboardNumber(),
        channels->getChannel(i)->isMappingActive(),
        channels->getChannel(i)->getMappedChannelIndex());
    }
}

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();
}

/*****HELPERS*****/
static string getLocalDateString()
{
    time_t timeObj;
    time(&timeObj);
    tm *pTime = localtime(&timeObj);
    char buffer[100];
    sprintf(buffer, "%d-%02d-%02d-%02d-%02d", pTime->tm_year + 1900, pTime->tm_mon + 1, pTime->tm_mday,
        pTime->tm_hour, pTime->tm_min, pTime->tm_sec);

    return buffer;
}

static vector<string> readZipsFromDirectory(string dir)
{
    vector<string> output;
    DIR* directory = nullptr;
    directory = opendir(dir.c_str());
    struct dirent* entry = readdir(directory);

    string file;
    while (entry != nullptr)
    {
        file = entry->d_name;

        if (file.find(".zip") != string::npos)
        {
            output.push_back(file);
        }

        entry = readdir(directory);
    }

    closedir(directory);
    return output;
}

static bool extractZipToDirectory(string zipPath, string destination)
{
    mz_zip_archive zip_archive;
    memset(&zip_archive, 0, sizeof(zip_archive));
    mz_bool status = mz_zip_reader_init_file(&zip_archive, zipPath.c_str(), 0);
    if (!status)
    {
        cout << "mz_zip_reader_init_file() failed!" << endl;
        return false;
    }
}

#ifdef _MSV_VER
    mkdir(destination.c_str());
#else
    mkdir(destination.c_str(), 0777);
#endif
for (size_t i = 0; i < (int)mz_zip_reader_get_num_files(&zip_archive); i++)
{

```

```
mz_zip_archive_file_stat file_stat;
if (!mz_zip_reader_file_stat(&zip_archive, i, &file_stat))
{
    cout << "mz_zip_reader_file_stat() failed!\n" << endl;
    mz_zip_reader_end(&zip_archive);
    return false;
}

size_t uncomp_size;
void* p = mz_zip_reader_extract_file_to_heap(&zip_archive, file_stat.m_filename, &uncomp_size, 0);
if (!p)
{
    printf("mz_zip_reader_extract_file_to_heap() failed!\n");
    mz_zip_reader_end(&zip_archive);
    return false;
}

ofstream outFile;
string fn = destination + file_stat.m_filename;
outFile.open(fn.c_str(), ios::binary);
if (!outFile.is_open())
{
    printf("Failed to write extracted file.\n");
    mz_zip_reader_end(&zip_archive);
    return false;
}

outFile.write((const char*)p, uncomp_size);
outFile.close();
}

mz_zip_reader_end(&zip_archive);
}
```

Chapter 2

Deprecated List

globalScope> Member [DEV_BP2](#)

For blue PiraT 2 devices use type *DEV_BP2_V1X*, for new blue PiraT 2 5E devices use *DEV_BP2_V2X*

Member [IBPNGClient::assignDBCFile](#) (int channelIndexCAN, const char *dbcFilePath)=0
, use function [assignDatabaseFile\(\)](#) instead

Member [IBPNGClient::clearDBCFileAssignments](#) ()=0
, use function [clearDatabaseFileAssignments\(\)](#) instead

Member [IBPNGClient::createNewConversionSet](#) ()=0
, use static function [createNewConversionSet\(\)](#) instead

Member [IBPNGClientListener::onProgressDataDownload](#) (int percentCompleted)=0
This function version is deprecated. Use the [onProgressDataDownload\(\)](#) with three arguments.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

BPNGError	25
DataSpan	32
IBPNGClient	32
IBPNGClientListener	50
BPNGLoggerDetector	25
IChannel	56
IChannelList	56
IClientProperties	57
IConversionSet	64
IFalseMeasureSignal	66
IFalseMeasureSignalList	66
IFormatInfo	66
IFormatList	67
IRdbEvent	67
IRdbEventList	68
IRdbTraceBlock	69
IRdbTraceBlockList	69
ITesttoolsChannel	69
ITesttoolsChannelList	70
LogInData	71
MemoryFillLevel	71
OnlineLoggerInfo	72
OnlineLoggerInfoStringPair	73
RdbEvent2	74
TSLCluster	75
vector	
RdbEventList	75

Chapter 4

Class Index

4.1 Class List

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

BPNGError	
Error struct with error code and optional error message	25
BPNGLoggerDetector	25
DataSpan	32
IBPNGClient	
Interface class for the Telemotive Client Library	32
IBPNGClientListener	50
IChannel	
Channel interface	56
IChannelList	
Channel list interface	56
IClientProperties	
The IClientProperties interface replaces the deprecated <i>ClientProperties</i> struct	57
IConversionSet	
A conversion set stores all conversion relevant settings	64
IFalseMeasureSignal	
False measure signal interface	66
IFalseMeasureSignalList	
False measure signal list interface	66
IFormatInfo	
FormatInfo interface	66
IFormatList	
Format list interface	67
IRdbEvent	
Interface to an RDB event	67
IRdbEventList	
Interface to a list of rdb events	68
IRdbTraceBlock	69
IRdbTraceBlockList	69
ITesttoolsChannel	
Channel interface	69
ITesttoolsChannelList	
TesttoolsChannel list interface	70
LoginData	
Structure for login	71

MemoryFillLevel	Stores memory fill level of a device	71
OnlineLoggerInfo	Struct with information about a logger found in LAN/WLAN used to notify IBP-NGClientListener about detected/disappeared devices	72
OnlineLoggerInfoStringPair	Helper object for configuration, license update or firmwareupdate: a key value pair for assigning a configuration, licensefile, etc. to a device	73
RdbEvent2	Implementation class for a wrapper of IRdbEvent using STL classes	74
RdbEventList	Implementation class for a wrapper of IRdbEventList using STL classes	75
TSLCluster	Representation of a chain of Telemotive devices combined via Telemotive System Link (TSL)	75

Chapter 5

File Index

5.1 File List

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

BPNGDefines.h	Defines for Telemotive Client Library	77
BPNGLoggerDetector.hh	Logger Detector Sample	86
IBPNGClient.h	Interface class for the BPNGClient DLL	86
IBPNGClientListener.h	Interface class for the BPNGClient listener	89
IClientProperties.h	Interface for client properties	89
RdbDefines.h	Public interfaces for Telemotive Reference Database access	90
RdbEventList.hh	IRdbEvent wrapper	91

Chapter 6

Class Documentation

6.1 BPNGError Struct Reference

Error struct with error code and optional error message.

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

Public Attributes

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

6.1.1 Detailed Description

Error struct with error code and optional error message.

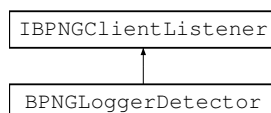
The documentation for this struct was generated from the following file:

- [BPNGDefines.h](#)

6.2 BPNGLoggerDetector Class Reference

```
#include <BPNGLoggerDetector.hh>
```

Inheritance diagram for BPNGLoggerDetector:



Public Member Functions

- [BPNGLoggerDetector](#) ()
- `std::vector< OnlineLoggerInfo >` [getLoggerList](#) (unsigned searchTimeOut)
- `std::vector< TSLClusterImpl >` [getTSLs](#) (`std::vector< OnlineLoggerInfo >` loggersInNetwork)

- 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 data transfer.
- virtual int WINAPI [onProgressDataDownload](#) (int percentCompleted, uint64_t downloaded-Size, uint64_t totalSize)
Called to indicate the current progress of a data transfer.
- virtual int WINAPI [onProgressConversion](#) (int percentCompleted, const char *status)
Called to indicate the current progress of file conversion.
- virtual int WINAPI [onProgressDeletion](#) (int percentCompleted)
Called to indicate the current progress of file deletion.
- virtual void WINAPI [onStatusMessage](#) (const char *statusMsg)
Called to send additional information of the current process to the calling app.
- virtual int WINAPI [onDataRecoverProgress](#) (const char *statusMsg, int percentage)
Called to send additional information of the current data recovery progress.
- 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 const char *WINAPI [onLogInDataRequired](#) (unsigned mbnr)
Called on accessing password protected functions.
- virtual void WINAPI [onInvalidPwConfigFound](#) (unsigned mbnr)
Called if invalid pw file found on device.
- virtual void WINAPI [onLogInDataFailed](#) ()
- virtual void WINAPI [onResetLogInDataFailed](#) ()
- virtual void WINAPI [onFuncAccessDenied](#) ()
- virtual int WINAPI [onCriticalDiskSpace](#) (uint64_t freeSpace, uint64_t neededSpace, const char *drive)
- 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)
- 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 [onDownloadStart](#) (int64_t totalAmountOfBytes)
Notifies the listeners before the download starts about the total amount of bytes to be downloaded.
- virtual void WINAPI [onConversionStart](#) (int64_t totalAmountOfBytes)
Notifies the listeners before the conversion starts about the total amount of bytes to be converted.
- virtual const char *WINAPI [onExtractionPasswordRequired](#) (unsigned int)
- virtual bool WINAPI [isTerminateLiveDownloadRequest](#) ()
Called periodically on live download to query whether the permanent download should be finished.

6.2.1 Detailed Description

A simple minimal implementation for a logger detection. Uses an [IBPNGClient](#) instance and the method [IBPNGClient::scanNetworkForLogger\(\)](#). Sets itself as [IBPNGClientListener](#) to the [IBPNGClient](#) instance ([IBPNGClient::addListener\(IBPNGClientListener* listener\)](#)). The callbacks [onBPNGDeviceDetected\(OnlineLoggerInfo *info\)](#), [onBPNGDeviceDisappeared\(OnlineLoggerInfo *info\)](#) and [onBPNGDeviceStateChange\(OnlineLoggerInfo *info\)](#) informs the class about the device states in network.

6.2.2 Constructor & Destructor Documentation

BPNGLoggerDetector::BPNGLoggerDetector () [inline]

Constructor

6.2.3 Member Function Documentation

std::vector<OnlineLoggerInfo> BPNGLoggerDetector::getLoggerList (unsigned *searchTimeOut*)

Returns a vector of detected BPNGDevice in local networks.

Parameters

<i>searchTimeOut</i>	the search timeout in seconds
----------------------	-------------------------------

Returns

vector of BPNGDevice

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](#).

std::vector<TSLClusterImpl> BPNGLoggerDetector::getTSLs (std::vector<OnlineLoggerInfo> *loggersInNetwork*)

Checks a vector of BPNGDevice for TSL chains. Combines the devices with same tslNetworkId (except -1) to [TSLCluster](#).

Parameters

<i>loggersInNetwork</i>	the BPNGDevice in network found by getLoggerList(unsigned searchTimeOut)
-------------------------	--

Returns

vector of [TSLCluster](#)

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](#).

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](#).

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](#).

virtual void WINAPI BPNGLoggerDetector::onConversionStart (*int64_t* *totalAmountOfBytes*) [inline], [virtual]

Notifies the listeners before the conversion starts about the total amount of bytes to be converted.
Parameters

<i>totalAmountOfBytes</i>	Total data size to be converted
---------------------------	---------------------------------

Implements [IBPNGClientListener](#).

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](#).

virtual int WINAPI BPNGLoggerDetector::onDataRecoverProgress (const char * *statusMsg*, int *percentage*) [inline], [virtual]

Called to send additional information of the current data recovery progress.

This function transmit message informations for the data recovery process. Those messages are only for information purpose. The information contains a String information about the current data recovery process and int value which contains a percent value for progressbar

Implements [IBPNGClientListener](#).

virtual void WINAPI BPNGLoggerDetector::onDownloadStart (int64_t *totalAmountOfBytes*) [inline], [virtual]

Notifies the listeners before the download starts about the total amount of bytes to be downloaded.

Parameters

<i>totalAmountOfBytes</i>	Total data size to be downloaded
---------------------------	----------------------------------

Implements [IBPNGClientListener](#).

virtual const char* WINAPI BPNGLoggerDetector::onExtractionPasswordRequired (unsigned *retryCount*) [inline], [virtual]

Notifies the listeners that a password for an archive extraction is required, this will be called on EVERY archive that needs a password nethertheless a password was already entered. Already entered passwords should be handled by the callbacked instance.

Parameters

<i>retryCount</i>	number of attempty on one file, on zero its first try The callbacked instance can save a password list and try every password on the list, if retryCount is zero the list should be handled from the start. If no password is left return 0.
-------------------	--

Implements [IBPNGClientListener](#).

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](#).

virtual void WINAPI BPNGLoggerDetector::onInvalidPwConfigFound (unsigned *mbnr*) [inline], [virtual]

Called if invalid pw file found on device.

An error may occure on transferring the password configuration to the device, as a result the password configuration is invalid and needs to be reset to default. Inform the user.

Implements [IBPNGClientListener](#).

virtual const char* WINAPI BPNGLoggerDetector::onLoginDataRequired (unsigned *mbnr*) [inline], [virtual]

Called on accessing password protected functions.

When password protected functions are called this listener function queries for login parameters that must be returned from the implementing class.

Parameters

<i>ipAddress</i>	IP address of the password protected device
------------------	---

Returns

Implementing class must return the username and password separated by a slash, e.g. "Tester/tge6ht". If an empty string is returned the login process will be aborted.

Implements [IBPNGClientListener](#).

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>percentCompleted</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](#).

virtual int WINAPI BPNGLoggerDetector::onProgressDataDownload (int *percentCompleted*) [inline], [virtual]

Called to indicate the current progress of a data transfer.

Deprecated This function version is deprecated. Use the [onProgressDataDownload\(\)](#) with three arguments.

This function notifies the listener about the download progress of the raw Telemotive trace data.

Parameters

<i>percentCompleted</i>	Percentage of the entire download process (from 0...100%). A negative value can be passed if only the abort request is checked. A negative value of -1 indicates a broken ftp connection.
-------------------------	---

Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

virtual int WINAPI BPNGLoggerDetector::onProgressDataDownload (int percentCompleted, uint64_t downloadedSize, uint64_t totalSize) [inline], [virtual]

Called to indicate the current progress of a data transfer.

This function notifies the listener about the download progress of the raw Telemotive trace data.

Parameters

<i>percentCompleted</i>	Percentage of the entire download process (from 0...100%). A negative value can be passed if only the abort request is checked. A negative value of -1 indicates a broken ftp connection.
<i>downloadedSize</i>	Amount of bytes already downloaded
<i>totalSize</i>	Total size to be downloaded

Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

virtual int WINAPI BPNGLoggerDetector::onProgressDeletion (int percentCompleted) [inline], [virtual]

Called to indicate the current progress of file deletion.

This function notifies the listener about the deletion progress of the raw Telemotive trace data.

Parameters

<i>percentCompleted</i>	Percentage of the entire deletion process (from 0...100%). A negative value can be passed if only the abort request is checked. A negative value of -1 indicates a broken ftp connection.
-------------------------	---

Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

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](#).

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 or the files of an offline data set 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. Returning another value than 1 will abort the current process with an error result.

Implements [IBPNGClientListener](#).

**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](#)

6.3 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
- uint64_t [reserved](#)

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

- [BPNGDefines.h](#)

6.4 IBPNGClient Struct Reference

Interface class for the Telemotive Client Library.

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

Public Member Functions

- virtual void WINAPI [scanNetworkForLogger](#) ()=0
Scan network for logger.
- virtual void WINAPI [activateGatewayLoggerDetection](#) ()=0
Activates the detection of devices connected to a different subnet.
- virtual void WINAPI [setDevice](#) (const [OnlineLoggerInfo](#) &device)=0
Sets the device, the [IBPNGClient](#) instance should operate with (download data, change configuration, update firmware, etc.)
- virtual void WINAPI [setTSLCluster](#) ([TSLCluster](#) cluster)=0
Sets the TSL cluster, the [IBPNGClient](#) instance should operate with (download data, change configuration, update firmware, etc.)
- virtual BOOL WINAPI [setOfflineData](#) (const char *path)=0
Sets the path to offline data set the [IBPNGClient](#) instance should operate with (conversion)
- virtual BOOL WINAPI [connect](#) ()=0
Connect to device or TSL set by [setDevice\(\)](#) or [setTSLCluster\(\)](#)
- virtual BOOL WINAPI [connectLogger](#) (int numLogger, [OnlineLoggerInfo](#) *devices)=0

- Connect to passed loggers.*

 - 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 void WINAPI [registerAll](#) ()=0

Static registration of all messages and formats.
- virtual BOOL WINAPI [initialize](#) ()=0

Initialization of download and conversion process.
- virtual int WINAPI [downloadDataSpans](#) (uint16_t numSpans, [DataSpan](#) *dataSpans, const char *target, BOOL doSorting)=0

Download trace data.
- virtual int WINAPI [startLiveDownload](#) (uint64_t startTimeStamp, const char *target)=0
- 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](#) (int numLogger, [OnlineLoggerInfoStringPair](#) *loggerConfigPathPairs, const char *xsdVersionOfConfig=nullptr)=0

Reconfig logger with the zipped new configuration.
- virtual BOOL WINAPI [setDefaultConfig](#) ()=0

Reconfig logger/TSL with the default configuration.
- virtual [IRdbEventList](#) *WINAPI [getEventList](#) ()=0

Get list of all events from the RDB.
- virtual [IRdbTraceBlockList](#) *WINAPI [getTraceBlockList](#) ()=0

Get list of all trace blocks from the RDB.
- virtual BOOL WINAPI [synchronizeRdb](#) ()=0

Synchronizes the RDB.
- virtual const char *WINAPI [getInstanceName](#) ()=0

Return the instance name passed to the [getBPNGClient\(\)](#) function.
- virtual int WINAPI [getInstanceId](#) ()=0

Returns the instance ID that is unique for all [IBPNGClient](#) instances in one process.
- 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 [ITesttoolsChannelList](#) *WINAPI [getLoggerTesttoolsChannels](#) ()=0
- virtual BOOL WINAPI [getVersions](#) ([OnlineLoggerInfoStringPair](#) *versionPairs)=0

Get the firmware and hardware version.
- virtual const char *WINAPI [getTMTVersion](#) ()=0

Get the current tmt version string.

- virtual BOOL WINAPI **updateFirmware** (OnlineLoggerInfoStringPair *loggerFirmwareUpdatePacketPair, BOOL force)=0
Update firmware.
- virtual BOOL WINAPI **updatePBFirmware** (OnlineLoggerInfoStringPair *loggerFirmwareUpdatePacketPair, BOOL force)=0
- virtual const char *WINAPI **getInternalPBVersion** (OnlineLoggerInfoStringPair *loggerFirmwareUpdatePacketPair)=0
- virtual BOOL WINAPI **isUserAuthenticated** (PwdPrivilegesFuncId actionName)=0
- virtual BOOL WINAPI **updateLicenses** (OnlineLoggerInfoStringPair *loggerLicenseFilePair)=0
Update licenses.
- virtual const char *WINAPI **getLicenses** (unsigned deviceMbnr)=0
Returns the license file's content of the specified device as string.
- virtual BOOL WINAPI **removeAllLicenses** ()=0
Removes the current license file from the logger.
- virtual int WINAPI **deleteSectionsByStartUplds** (uint16_t numStartUplds, uint64_t *startupUplds)=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 int WINAPI **getCurrentLoggerTime** ()=0
Returns the current loggertime in seconds since 01.01.1970 UTC.
- virtual int WINAPI **setTime** (int 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 OnlineLoggerInfo *WINAPI **getOnlineLoggerInfo** (const char *ip)=0
Retreive an OnlineLoggerInfo for a particular IP address.
- 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 **createCCPXCPSeqFile** (const char *xsdDir, const char *xmlDir, bool forceFlag)=0
Parse CCPXCPMeasurement.xml and CCPXCPCConfiguration.xml and create CCPXCPSquence.xml.
- virtual int WINAPI **createCCPXCPDbcFiles** (const char *dbcDir, const char *xsdDir, const char *xmlDir)=0
Parse CCPXCPMeasurement.xml and CCPXCPCConfiguration.xml and create a Vector DBC file for each device.
- virtual const IFalseMeasureSignalList *WINAPI **getFalseMeasureSignals** ()=0
Return pointer to a false measure signal list interface.
- 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 `getFWVersion` ()=0

Returns the current fw version.
- virtual `void` WINAPI `release` ()=0

Free memory of this `IBPNGClient` instance.
- virtual `IClientProperties *`WINAPI `getClientProperties` ()=0
- virtual `void` WINAPI `setClientProperties` (`IClientProperties *`properties)=0
- virtual `void` WINAPI `saveProperties` (const char *pathToIniFile)=0

Save properties to ini file.
- virtual `void` WINAPI `loadProperties` (const char *pathToIniFile)=0

Load properties from ini file.
- virtual `void` WINAPI `clearDBCFileAssignments` ()=0

Remove all DBC file assignments.
- virtual `void` WINAPI `clearDatabaseFileAssignments` ()=0

Remove all database file assignments.
- virtual `void` WINAPI `assignDBCFile` (int channelIndexCAN, const char *dbcFilePath)=0

Assign a DBC file to a CAN channel. Multiple files for one CAN channel are allowed, but double used message IDs will ignored.
- virtual `void` WINAPI `assignDatabaseFile` (`ChannelType` channelType, int channelIndex, const char *databaseFilePath)=0

Assign a database file to a channel. Multiple files for one channel are allowed, but double used message IDs will ignored.
- virtual `int` WINAPI `resetMarkerCounter` ()=0

Reset marker counter.
- virtual `int` WINAPI `setPwdFile` (const char *path, unsigned targetMbnr)=0
- virtual `const char *`WINAPI `getPwdFile` (unsigned sourceMbnr)=0
- virtual `BOOL` WINAPI `isPasswordProtectionSupported` (unsigned deviceMbnr)=0
- virtual `int` WINAPI `downloadBugReport` (const char *targetPath, `BPNGBugreportMode` mode, uint64_t startTime, uint64_t endTime)=0

Download bug report.
- virtual `int` WINAPI `restartDevice` (`BOOL` waitForRestart)=0

restarts the device or TSL
- virtual `int` WINAPI `shutdownDevice` ()=0

shut down the device or TSL
- virtual `BOOL` WINAPI `getMemoryFillLevel` (`MemoryFillLevel` *fillLevel)=0

get memory fill level of device
- virtual `BOOL` WINAPI `convertFileNameTimeStampsToLocalTime` (const char *pathToOfflineDataSet)=0

Converts all time stamps in an offline data set's file names to local time.

- virtual BOOL WINAPI [filterSignals](#) (const char *pathToFilterSettings, const char *targetPath)=0
Signal filtering.
- virtual BOOL WINAPI [filterSignalsFromOfflineData](#) (const char *pathToOfflineDataSet, const char *pathToFilterSettings, const char *targetPath)=0
Signal filtering.
- virtual int WINAPI [createTestReport](#) (IConversionSet *conversionSet, const char *target)=0
easy track test report creation
- virtual void WINAPI [enableClientLogOutput](#) ()=0
enable ClientLogUtil output

6.4.1 Detailed Description

Interface class for the Telemotive Client Library.

[IBPNGClient](#) is the interface class of the blue PiraT Client library. To get access to any Telemotive data logger except "blue PiraT 1" 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 conflicts 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.

To get access to a device chain combined via Telemotive System Link (TSL) you also need a pointer to an implementing instance of [IBPNGClient](#) interface. Since version 4.1.1 you can use the same factory function [getBPNGClient\(\)](#). The prior method [getTSLClient\(int numTSLMember\)](#) is no longer available. Please see the documentation of the new methods [setDevice\(\)](#), [setTSLCluster\(\)](#) and [setOfflineData\(\)](#) regarding this issue.

6.4.2 Member Function Documentation

virtual void WINAPI IBPNGClient::activateGatewayLoggerDetection () [pure virtual]

Activates the detection of devices connected to a different subnet.

Logger connected to a different subnet must be configured as DHCP client and must have enabled the PingToClient option to be detectable via the [scanNetworkForLogger\(\)](#) function.

Calling this function will start a background thread that waits for incoming pings of such devices.

virtual void WINAPI IBPNGClient::assignDatabaseFile (ChannelType channelType, int channelIndex, const char * databaseFilePath) [pure virtual]

Assign a database file to a channel. Multiple files for one channel are allowed, but double used message IDs will ignored.

Supported database formats and channelTypes: CH_CAN (also CAN-FD): FIBEX, DBC, Autosar XML CH_FLEXRAY: FIBEX, Autosar XML CH_MII: Autosar XML Supported FIBEX versions: 3.1.0, 3.1.1, 4.0.0, 4.1.0, 4.1.1 Supported Autosar XML versions: 3.1.4.A1.4, 3.2.1, 3.2.2, 3.2.3, r4.0

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

Assign a DBC file to a CAN channel. Multiple files for one CAN channel are allowed, but double used message IDs will ignored.

Deprecated , use function [assignDatabaseFile\(\)](#) instead

will be removed with next vesion!

Parameters

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

virtual void WINAPI IBPNGClient::clearDBCFileAssignments () [pure virtual]

Remove all DBC file assignments.

Deprecated , use function [clearDatabaseFileAssignments\(\)](#) instead

will be removed with next vesion!

virtual BOOL WINAPI IBPNGClient::connectLogger (int *numLogger*, OnlineLoggerInfo * *devices*) [pure virtual]

Connect to passed loggers.

While the loggers are connected, they 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\(\)](#).

Parameters

<i>numLogger</i>	the number of passed OnlineLoggerInfo devices
<i>devices</i>	pointer to first OnlineLoggerInfo

Returns

0 on failure, 1 on success

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::initialize\(\)](#) 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 IConversionSet
<i>target</i>	target directory for the converted trace files. Depending on the passed Client-Properties the files may be stored in sub folders named by date

Returns

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

virtual IConversionSet* WINAPI IBPNGClient::createNewConversionSet () [pure virtual]

Returns the pointer to a new conversion set.

Deprecated , use static function [createNewConversionSet\(\)](#) instead

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

easy track test report creation

This function creates test reports for every section started by START_TESTDRIVE and ended by END_TESTDRIVE. Every test report will get a own folder in the target directory, every timespan on conversionset is a failure and gets it own folder in the test report folder. All selected data in the conversion set will be converted in that failure folder.

Parameters

<i>conversionSet</i>	conversion settings, see IConversionSet
<i>target</i>	target directory for the test reports.

Returns

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

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.

virtual int WINAPI IBPNGClient::deleteSectionsByStartUplds (uint16_t *numStartUplds*, uint64_t * *startUplds*) [pure virtual]

Delete trace data.

Pass the size and the pointer to an array of RDB-DataBaseEntryIDs all from RdbEvents with RdbEventType 'STARTUP'. If you want to retrieve the section startUplds you have to call the [initialize\(\)](#) 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\(\)](#). Note that this function is not available for blue PiraT Collect devices (BPNGDeviceType DEV_TRACE_COLL_DS).

Parameters

<i>numStartUplds</i>	Size of the passed uint64_t array in second parameter
<i>startUplds</i>	Array of uint64_t, specifying the startUplds of the sections that should be deleted

Returns

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

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.

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::initialize\(\)](#) 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 DataSpan array in second parameter
<i>dataSpans</i>	Array of DataSpan , 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.

virtual void WINAPI IBPNGClient::enableClientLogOutput () [pure virtual]

enable ClientLogUtil output

This function enable the output of class ClientLogUtil.

virtual BOOL WINAPI IBPNGClient::filterSignals (const char * *pathToFilterSettings*, const char * *targetPath*) [pure virtual]

Signal filtering.

This function parses all data of the offline data set that was previously set via [setOfflineData\(\)](#) and filters signals according to the complex filter settings created with the System Client.

Parameters

<i>pathToFilter-Setting</i>	path to the ZIP file including the complex filter settings created with System Client
<i>targetPath</i>	path to the target directory where the filtered data should be written to

virtual BOOL WINAPI IBPNGClient::filterSignalsFromOfflineData (const char * *pathToOfflineDataSet*, const char * *pathToFilterSettings*, const char * *targetPath*) [pure virtual]

Signal filtering.

This function parses all data of an offline data set and filters signals according to the complex filter settings created with the System Client.

Parameters

<i>pathToOffline-DataSet</i>	path to the offline data set
<i>pathToFilter-Setting</i>	path to the ZIP file including the complex filter settings created with System Client
<i>targetPath</i>	path to the target directory where the filtered data should be written to

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. On TSL all device LEDs will flash.

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](#)

virtual IClientProperties* WINAPI IBPNGClient::getClientProperties () [pure virtual]

See also

[IClientProperties](#), [setClientProperties\(\)](#)

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.

On TSL instance you have to pass a base path. All participants logger configurations will be saved as zip in that directory including a TSLConfig.txt file with additional informations.

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 or on TSL the basepath for config zips
-------------	--

virtual const char* WINAPI IBPNGClient::getConfigPath () [pure virtual]

Get path to the config directory (after calling one of the init functions)

After calling [IBPNGClient::initialize\(\)](#) this function returns the path to the current extracted configuration of the logger resp. the offline data set. **On TSL instance you get paths to the config folders of every participant separated by ';': <configpath:1>;<configPath:2>;... for example C:..BLUEPIRAT\PNGINST25452\BPTSL_10.64.76.171_2\BP2Img_10.64.76.171_3_PID5452;C:..BLUEPIRA**

Returns

Path to the folder containing the extracted config archive.

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.

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

Get name of device.

After calling [IBPNGClient::initialize\(\)](#) this function returns the currently configured device name. On TSL the device names will be separated by ';'.

Returns

The device name

See also

[initialize\(\)](#)

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.

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

Get list of all events from the RDB.

The events of the device's or offline data set's RDB is returned. [IBPNGClient::initialize\(\)](#) must have been called before

Returns

Pointer to a [IRdbEventList](#)

virtual const IFalseMeasureSignalList* WINAPI IBPNGClient::getFalseMeasureSignals () [pure virtual]

Return pointer to a false measure signal list interface.

After calling the [IBPNGClient::createCCPXCPCDbcFiles\(\)](#) this function returns a pointer to a list with all measure signals which were ignored at DBC file generation.

See also

[IFalseMeasureSignal](#)

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](#)

virtual const char* WINAPI IBPNGClient::getLicenses (unsigned *deviceMbnr*) [pure virtual]

Returns the license file's content of the specified device as string.

Parameters

<i>deviceMbnr</i>	target device mainboardnumber
-------------------	-------------------------------

Returns

license file's content as string

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

Returns pointer to a channel list interface.

After calling [IBPNGClient::initialize\(\)](#) this function returns a pointer to the logger's/TSL 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](#)

virtual BOOL WINAPI IBPNGClient::getMemoryFillLevel (MemoryFillLevel * fillLevel) [pure virtual]

get memory fill level of device

On TSL ensure the fillLevel structure has reserved enough space for all members

Parameters

<i>fillLevel</i>	structure description in bpngdefines
------------------	--

Returns

0 on failure, 1 on success

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\(\)](#)

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\(\)](#)

virtual const char* WINAPI IBPNGClient::getPwdFile (unsigned sourceMbnr) [pure virtual]

get the password file of device specified by the mainboardnumber

Parameters

<i>sourceMbnr</i>	the source device mainboardnumber
-------------------	-----------------------------------

Returns

local path to file

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

Get path to the reference data base.

After calling [IBPNGClient::initialize\(\)](#) 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

[initialize\(\)](#)

virtual IRdbTraceBlockList* WINAPI IBPNGClient::getTraceBlockList () [pure virtual]

Get list of all trace blocks from the RDB.

Returns

Pointer to a [IRdbEventList](#)

virtual BOOL WINAPI IBPNGClient::getVersions (OnlineLoggerInfoStringPair * *versionPairs*) [pure virtual]

Get the firmware and hardware version.

On TSL ensure the versionPairs structure has reserved enough space for all members!
the versionPairs.value will be the firmware and version string the versionPairs.key.mbnr will be the referenced device. Only the mbnr field will be filled, the other fields will be empty!

Parameters

<i>versionPairs</i>	structure description in bpngdefines
---------------------	--------------------------------------

Returns

0 on failure, 1 on success

virtual BOOL WINAPI IBPNGClient::initialize () [pure virtual]

Initialization of download and conversion process.

For trace download and conversion this function must be called first. When operating on a network device, note you have to call the [connect\(\)](#) function before.

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

virtual BOOL WINAPI IBPNGClient::isPasswordProtectionSupported (unsigned deviceMbnr) [pure virtual]

check if the device supports password protection

Parameters

<i>deviceMbnr</i>	the device
-------------------	------------

Returns

1 if the device supports password protection

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\(\)](#)

virtual BOOL WINAPI IBPNGClient::reconfigLogger (int numLogger, OnlineLoggerInfoStringPair * loggerConfigPathPairs, const char * xsdVersionOfConfig = nullptr) [pure virtual]

Reconfig logger with the zipped new configuration.

Reconfigures the logger/tsl with the passed configurations. A single device can be reconfigured with a ZIP archive downloaded with the [getConfig\(\)](#) method or stored by the client software. In case of TSL cluster [getConfig\(\)](#) and the client software stores a ZIP which contains several ZIPs - one for each device. Those TSL configuration ZIPs can not be applied directly to a TSL cluster. You have to extract them and must assign each of the single device configurations to the appropriate [OnlineLoggerInfo](#).

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. The filename must include the current date in followed form: [YYYY-MM-DD_HH-MM-SS] -> Y=year, M=month, D=day, H=hour, M=minute, S=second

With the [OnlineLoggerInfoStringPair](#) structure you can assign the several configurations to the devices. [OnlineLoggerInfoStringPair.key](#) = [OnlineLoggerInfo](#) [OnlineLoggerInfoStringPair.value](#) = path to local config file

See also

[OnlineLoggerInfoStringPair](#)

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>numLogger</i>	Number of following OnlineLoggerInfoStringPair (should be equal to the number of devices on TSL)
<i>loggerToConfigPathPairs</i>	Pointer to first OnlineLoggerInfoStringPair
<i>xsdVersionOfConfig</i>	xsd version of config version, required if the version differs from the logger config version. Requires FW 4.1.1 or higher. The xsd version can be found in the FirmwareConfiguration.xml.

Returns

0 on failure, 1 on success

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!

virtual BOOL WINAPI IBPNGClient::removeAllLicenses () [pure virtual]

Removes the current license file from the logger.

Removes the current license file from the logger.

Returns

true on success, false on failure

virtual int WINAPI IBPNGClient::restartDevice (BOOL *waitForRestart*) [pure virtual]

restarts the device or TSL

Parameters

<i>waitForRestart</i>	if 1 communication waits for the restart
-----------------------	--

Returns

0 on failure, 1 on success, -1 on false fw version

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 600ms 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.

virtual void WINAPI IBPNGClient::setClientProperties (IClientProperties * *properties*)
[pure virtual]

Parameters

<i>Pointer</i>	to IClientProperties which can be retrieved from the static function createNewClientProperties() or from IBPNGClient::getClientProperties()
----------------	---

See also

[IClientProperties](#), [getClientProperties\(\)](#), [createNewClientProperties](#)

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

Reconfig logger/TSL with the default configuration.

An invalid configuration will set the logger/TSL in error state. To fix this one possibility is to set the logger's default configuration. On TSL every logger will be reset to default configuration.

Returns

0 on failure, 1 on success

virtual void WINAPI IBPNGClient::setDevice (const OnlineLoggerInfo & *device*) [pure virtual]

Sets the device, the [IBPNGClient](#) instance should operate with (download data, change configuration, update firmware, etc.)

Since Lib version 4.1.1 the proceeding who to connect to a device changed. Instead of passing the device resp. its IP address you want to operate with to the [connect\(\)](#) function, the device must now be set with this function before calling [connect\(\)](#). If you don't use an [OnlineLoggerInfo](#) received via [IBPNGClientListener::onBPNGDeviceDetected](#), only the IP parameter of [OnlineLoggerInfo](#) is obligatory. Example:

```
IBPNGClient* client = getBPNGClient();
OnlineLoggerInfo device = createEmptyOnlineLoggerInfo();
device.ip = "192.168.0.233";
client->setDevice(device);
client->connect();
```

See also

[setTSLCluster](#), [setOfflineData](#)

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

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. On TSL the marker will be broadcasted internally.

Returns

Returns 0 on failure, 1 on success

virtual BOOL WINAPI IBPNGClient::setOfflineData (const char * *path*) [pure virtual]

Sets the path to offline data set the [IBPNGClient](#) instance should operate with (conversion)

Since Lib version 4.1.1 the proceeding who to process offline data changed. Instead of passing the file path directly to the `initOnline()` function, the file path must now be set with this function before calling `initialize()`.

virtual int WINAPI IBPNGClient::setPwdFile (const char * *path*, unsigned *targetMbnr*)
[pure virtual]

set the password file on device specified by the mainboardnumber

Parameters

<i>path</i>	local path of password file
<i>targetMbnr</i>	the target device mainboardnumber

Returns

0 on failure, 1 on success

virtual int WINAPI IBPNGClient::setTime (int *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. On TSL the new time will be applied on every device.

Returns

-1 on clientLib busy, 0 on failure, 1 on success

virtual void WINAPI IBPNGClient::setTSLCluster (TSLCluster *cluster*) [pure virtual]

Sets the TSL cluster, the [IBPNGClient](#) instance should operate with (download data, change configuration, update firmware, etc.)

Since Lib version 4.1.1 the proceeding who to connect to a TSL changed. Instead of passing the devices directly to the [connect\(\)](#) function, the TSL must now be set with this function before calling [connect\(\)](#). You can use objects of TSLClusterImpl provided with the library package or create your own [TSLCluster](#) instance. Example:

```
IBPNGClient* client = getBPNGClient();
OnlineLoggerInfo dev1 = createEmptyOnlineLoggerInfo();
dev1.ip = "10.23.224.178";
OnlineLoggerInfo dev2 = createEmptyOnlineLoggerInfo();
dev2.ip = "10.23.224.56";
TSLClusterImpl cluster;
cluster.addDevice(dev1);
cluster.addDevice(dev2);
client->setTSLCluster(cluster.getTSLCluster());
client->connect();
```

virtual int WINAPI IBPNGClient::shutdownDevice () [pure virtual]

shut down the device or TSL

Returns

0 on failure, 1 on success, -1 on false fw version

virtual int WINAPI IBPNGClient::startLiveDownload (uint64_t *startTimeStamp*, const char * *target*) [pure virtual]

Downloads all available trace data younger than *startTimeStamp* from connected device and continues downloading all new captured traces periodically until an added listener returns zero in the implementation of IBNGClientListener::onDownloadProgress().

Parameters

<i>startTimeStamp</i>	Traces younger than this time stamp will be downloaded
<i>target</i>	Target path where offline data should be stored. Can be a directory or ZIP archive. If directory or ZIP name contains placeholder 'ENDTIME', the file/folder name will be renamed when download was stopped by replacing 'ENDTIME' with the maximum downloaded trace time stamp

virtual BOOL WINAPI IBPNGClient::synchronizeRdb () [pure virtual]

Synchronizes the RDB.

After calling [initialize\(\)](#) once you can use this function to synchronize the RDB that [getEventList\(\)](#) and [getTraceBlockList\(\)](#) will return the updated lists.

virtual BOOL WINAPI IBPNGClient::updateFirmware (OnlineLoggerInfoStringPair * *loggerFirmwareUpdatePacketPair*, 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 then the component's version inside the firmware packet will be updated.

Parameters

<i>loggerToFirmwareUpdatePacketPair</i>	A pair with key= OnlineLoggerInfo , the device to be updated and value=local path to firmware packet
<i>force</i>	Flag whether to update the components independently from the components' versions

Returns

0 on failure, 1 on success

virtual BOOL WINAPI IBPNGClient::updateLicenses ([OnlineLoggerInfoStringPair](#) * *loggerLicenseFilePair*) [pure virtual]

Update licenses.

Overwrites the current license file with the new one.

Parameters

<i>loggerLicenseFilePair</i>	A pair with key= OnlineLoggerInfo , the device to be updated and value=local path to license file
------------------------------	---

Returns

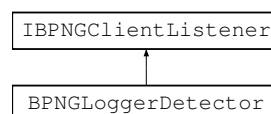
0 on failure, 1 on success

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

- [IBPNGClient.h](#)

6.5 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 data transfer.

- virtual int WINAPI [onProgressDataDownload](#) (int percentCompleted, uint64_t downloaded-Size, uint64_t totalSize)=0
Called to indicate the current progress of a data transfer.
- virtual int WINAPI [onProgressConversion](#) (int percentCompleted, const char *status)=0
Called to indicate the current progress of file conversion.
- virtual int WINAPI [onProgressDeletion](#) (int percentCompleted)=0
Called to indicate the current progress of file deletion.
- virtual void WINAPI [onStatusMessage](#) (const char *statusMsg)=0
Called to send additional information of the current process to the calling app.
- virtual int WINAPI [onDataRecoverProgress](#) (const char *statusMsg, int percentage)=0
Called to send additional information of the current data recovery progress.
- 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 const char *WINAPI [onLogInDataRequired](#) (unsigned mbnr)=0
Called on accessing password protected functions.
- virtual void WINAPI [onInvalidPwConfigFound](#) (unsigned mbnr)=0
Called if invalid pw file found on device.
- virtual void WINAPI [onLogInDataFailed](#) ()=0
- virtual void WINAPI [onResetLogInDataFailed](#) ()=0
- virtual void WINAPI [onFuncAccessDenied](#) ()=0
- 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
- virtual void WINAPI [onDownloadStart](#) (int64_t totalAmountOfBytes)=0
Notifies the listeners before the download starts about the total amount of bytes to be downloaded.
- virtual void WINAPI [onConversionStart](#) (int64_t totalAmountOfBytes)=0
Notifies the listeners before the conversion starts about the total amount of bytes to be converted.
- virtual const char *WINAPI [onExtractionPasswordRequired](#) (unsigned int retryCount)=0
- virtual bool WINAPI [isTerminateLiveDownloadRequest](#) ()=0
Called periodically on live download to query whether the permanent download should be finished.

6.5.1 Member Function Documentation

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](#).

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](#).

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](#).

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](#).

virtual void WINAPI IBPNGClientListener::onConversionStart (`int64_t` *totalAmountOfBytes*) [pure virtual]

Notifies the listeners before the conversion starts about the total amount of bytes to be converted.

Parameters

<i>totalAmountOfBytes</i>	Total data size to be converted
---------------------------	---------------------------------

Implemented in [BPNGLoggerDetector](#).

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](#).

virtual int WINAPI IBPNGClientListener::onDataRecoverProgress (const char * *statusMsg*, int *percentage*) [pure virtual]

Called to send additional information of the current data recovery progress.

This function transmit message informations for the data recovery process. Those messages are only for information purpose. The information contains a String information about the current data recovery process and int value which contains a percent value for progressbar

Implemented in [BPNGLoggerDetector](#).

virtual void WINAPI IBPNGClientListener::onDownloadStart (int64_t *totalAmountOfBytes*) [pure virtual]

Notifies the listeners before the download starts about the total amount of bytes to be downloaded.

Parameters

<i>totalAmountOfBytes</i>	Total data size to be downloaded
---------------------------	----------------------------------

Implemented in [BPNGLoggerDetector](#).

virtual const char* WINAPI IBPNGClientListener::onExtractionPasswordRequired (unsigned int *retryCount*) [pure virtual]

Notifies the listeners that a password for an archive extraction is required, this will be called on EVERY archive that needs a password nethertheless a password was already entered. Already entered passwords should be handled by the callbacked instance.

Parameters

<i>retryCount</i>	number of attempty on one file, on zero its first try The callbacked instance can save a password list and try every password on the list, if retryCount is zero the list should be handled from the start. If no password is left return 0.
-------------------	--

Implemented in [BPNGLoggerDetector](#).

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](#).

virtual void WINAPI IBPNGClientListener::onInvalidPwConfigFound (unsigned *mbnr*) [pure virtual]

Called if invalid pw file found on device.

An error may occure on transferring the password configuration to the device, as a result the password configuration is invalid and needs to be reset to default. Inform the user.

Implemented in [BPNGLoggerDetector](#).

virtual const char* WINAPI IBPNGClientListener::onLoginDataRequired (unsigned *mbnr*) [pure virtual]

Called on accessing password protected functions.

When password protected functions are called this listener function queries for login parameters that must be returned from the implementing class.

Parameters

<i>ipAddress</i>	IP address of the password protected device
------------------	---

Returns

Implementing class must return the username and password separated by a slash, e.g. "Tester/tge6ht". If an empty string is returned the login process will be aborted.

Implemented in [BPNGLoggerDetector](#).

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>percentCompleted</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](#).

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

Called to indicate the current progress of a data transfer.

Deprecated This function version is deprecated. Use the [onProgressDataDownload\(\)](#) with three arguments.

This function notifies the listener about the download progress of the raw Telemotive trace data.

Parameters

<i>percentCompleted</i>	Percentage of the entire download process (from 0...100%). A negative value can be passed if only the abort request is checked. A negative value of -1 indicates a broken ftp connection.
-------------------------	---

Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

virtual int WINAPI IBPNGClientListener::onProgressDataDownload (int *percentCompleted*, uint64_t *downloadedSize*, uint64_t *totalSize*) [pure virtual]

Called to indicate the current progress of a data transfer.

This function notifies the listener about the download progress of the raw Telemotive trace data.

Parameters

<i>percentCompleted</i>	Percentage of the entire download process (from 0...100%). A negative value can be passed if only the abort request is checked. A negative value of -1 indicates a broken ftp connection.
<i>downloadedSize</i>	Amount of bytes already downloaded
<i>totalSize</i>	Total size to be downloaded

Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

virtual int WINAPI IBPNGClientListener::onProgressDeletion (int *percentCompleted*) [pure virtual]

Called to indicate the current progress of file deletion.

This function notifies the listener about the deletion progress of the raw Telemotive trace data.

Parameters

<i>percentCompleted</i>	Percentage of the entire deletion process (from 0...100%). A negative value can be passed if only the abort request is checked. A negative value of -1 indicates a broken ftp connection.
-------------------------	---

Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

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](#).

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 or the files of an offline data set 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. Returning another value than 1 will abort the current process with an error result.

Implemented in [BPNGLoggerDetector](#).

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](#)

6.6 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.
- virtual uint32_t [getMainboardNumber](#) () const =0
Returns the channel's package id.
- virtual uint32_t [getOffset](#) () const =0
Returns the channel's offset.
- virtual BOOL [isMappingActive](#) () const =0
Returns whether the channel is mapped.
- virtual uint8_t [getMappedChannelIndex](#) () const =0
Returns the channel's mapped channel index.

6.6.1 Detailed Description

Channel interface.

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

- [BPNGDefines.h](#)

6.7 IChannelList Struct Reference

Channel list interface.

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

Public Member Functions

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

6.7.1 Detailed Description

Channel list interface.

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

- [BPNGDefines.h](#)

6.8 IClientProperties Struct Reference

The [IClientProperties](#) interface replaces the deprecated *ClientProperties* struct.

```
#include <IClientProperties.h>
```

Public Member Functions

- virtual void WINAPI [setCommonProperties](#) (const char *nameOfTester, int maxOutputSizeMB, BOOL separatedTimeFormat, BOOL separatedTimeFormatInOfflineSet, const char *alternativeLoggerName, BOOL useAlternativeLoggerName, BOOL useSubDirectories, BOOL midnightSplitting, BOOL fileTimeSpansLikeSelection, BOOL markerNumberInFileNames, BOOL subfolderWithLoggerName, int maxOfflineZipSizeMB, int maxOutputSizeMBSortedDownload, BOOL traceCutterStorage, const char *traceCutterFNPatternConversion, BOOL createOfflineDataOnTraceCutterStorage, const char *traceCutterFNPatternOfflineData, BOOL traceCutterMarkerCompact, BOOL pauseDataRecordingDuringDownload, BOOL tmASCIITimestampPrecisionToMicros)=0
Set Common properties.
- virtual void WINAPI [setNameOfTester](#) (const char *name)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setMaxOutputSize](#) (int size)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setSeparatedTimeFormat](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setSeparatedTimeFormatInOfflineSet](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setAlternativeLoggerName](#) (const char *name)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setAlternativeLoggerNameActive](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setConvertedFilesInSubDirsActive](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setMidnightSplittingActive](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setFileTimeSpansLikeSelection](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)

- virtual void WINAPI [setMarkerNumbersInFileNames](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setSubfolderWithLoggerName](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setMaxOfflineZipSize](#) (int size)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setMaxOutputSizeSortedDownload](#) (int size)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setTraceCutterStorage](#) (BOOL flag)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI [setTraceCutterFNPatternConversion](#) (const char *pattern)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI **setCreateOfflineDataOnTraceCutterStorage** (BOOL flag)=0
- virtual void WINAPI [setTraceCutterFNPatternOfflineData](#) (const char *pattern)=0
see parameter description of [setCommonProperties\(\)](#)
- virtual void WINAPI **setTraceCutterMarkerCompact** (BOOL flag)=0
- virtual void WINAPI **setPauseDataRecordingDuringDownload** (BOOL flag)=0
- virtual void WINAPI **setTMASCIITimestampPrecisionToMicros** (BOOL flag)=0
- virtual const char *WINAPI [getNameOfTester](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual int WINAPI [getMaxOutputSize](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isSeparatedTimeFormat](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isSeparatedTimeFormatInOfflineSet](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual const char *WINAPI [getAlternativeLoggerName](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isAlternativeLoggerNameActive](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isConvertedFilesInSubDirsActive](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isMidnightSplittingActive](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isFileTimeSpansLikeSelection](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isMarkerNumbersInFileNames](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isSubfolderWithLoggerName](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual int WINAPI [getMaxOfflineZipSize](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual int WINAPI [getMaxOutputSizeSortedDownload](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual BOOL WINAPI [isTraceCutterStorage](#) ()=0
see parameter description of [setCommonProperties\(\)](#)
- virtual const char *WINAPI [getTraceCutterFNPatternConversion](#) ()=0

- see parameter description of [setCommonProperties\(\)](#)*
- virtual BOOL WINAPI **isCreateOfflineDataOnTraceCutterStorage** ()=0
- virtual const char *WINAPI **getTraceCutterFNPatternOfflineData** ()=0
 - see parameter description of [setCommonProperties\(\)](#)*
- virtual BOOL WINAPI **isTraceCutterMarkerCompact** ()=0
- virtual BOOL WINAPI **isPauseDataRecordingDuringDownload** ()=0
- virtual BOOL WINAPI **isTMASCIITimestampPrecisionToMicros** ()=0
- virtual void WINAPI **setCANPseudoMsgTimeStampProperties** (BOOL writeTimeStampMsg, uint32_t channelIndex, uint32_t dlc, uint32_t canID, uint32_t hourBitPos, uint32_t minBitPos, uint32_t secBitPos, uint32_t dayBitPos, uint32_t monthBitPos, uint32_t yearBitPos)=0
 - Set CAN pseudo properties for writing time stamp messages.*
- virtual BOOL WINAPI **isCANPseudoMsgTimeStampActive** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgChannelIndexTimeStamp** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgDlcTimeStamp** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgCanIDTimeStamp** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgHourBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgMinBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgSecBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgDayBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgMonthBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgYearBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTimeStampProperties\(\)](#)*
- virtual void WINAPI **setCANPseudoMsgTriggerProperties** (BOOL writeTriggerMessage, uint32_t channelIndex, uint32_t dlc, uint32_t canID, uint32_t triggerNumBitPos)=0
 - Set CAN pseudo properties for writing trigger messages.*
- virtual BOOL WINAPI **isCANPseudoMsgTriggerActive** ()=0
 - see parameter description of [setCANPseudoMsgTriggerProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgChannelIndexTrigger** ()=0
 - see parameter description of [setCANPseudoMsgTriggerProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgDlcTrigger** ()=0
 - see parameter description of [setCANPseudoMsgTriggerProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgCanIDTrigger** ()=0
 - see parameter description of [setCANPseudoMsgTriggerProperties\(\)](#)*
- virtual uint32_t WINAPI **getCANPseudoMsgTriggerNumBitPos** ()=0
 - see parameter description of [setCANPseudoMsgTriggerProperties\(\)](#)*
- virtual void WINAPI **setMOSTPseudoMsgProperties** (BOOL active, uint32_t src, uint32_t target, uint32_t fktBlockID, uint32_t fktID)=0
 - Set MOST pseudo properties.*

- virtual BOOL WINAPI **isMOSTPseudoMsgActive** ()=0
see parameter description of [setMOSTPseudoMsgProperties\(\)](#)
- virtual uint32_t WINAPI **getMOSTPseudoMsgSourceAddr** ()=0
see parameter description of [setMOSTPseudoMsgProperties\(\)](#)
- virtual uint32_t WINAPI **getMOSTPseudoMsgTargetAddr** ()=0
see parameter description of [setMOSTPseudoMsgProperties\(\)](#)
- virtual uint32_t WINAPI **getMOSTPseudoMsgFktBlockID** ()=0
see parameter description of [setMOSTPseudoMsgProperties\(\)](#)
- virtual uint32_t WINAPI **getMOSTPseudoMsgFktID** ()=0
see parameter description of [setMOSTPseudoMsgProperties\(\)](#)
- virtual void WINAPI **setFlexRayPseudoMsgProperties** (BOOL active, uint8_t channel, uint32_t slotID, uint32_t cycleCount, BOOL useFlexRayTypeDynamic)=0
Set FlexRay pseudo properties.
- virtual BOOL WINAPI **isFlexRayPseudoMsgActive** ()=0
see parameter description of [setFlexRayPseudoMsgProperties\(\)](#)
- virtual uint8_t WINAPI **getFlexRayPseudoMsgChannel** ()=0
see parameter description of [setFlexRayPseudoMsgProperties\(\)](#)
- virtual uint32_t WINAPI **getFlexRayPseudoSlotID** ()=0
see parameter description of [setFlexRayPseudoMsgProperties\(\)](#)
- virtual uint32_t WINAPI **getFlexRayPseudoCycleCount** ()=0
see parameter description of [setFlexRayPseudoMsgProperties\(\)](#)
- virtual BOOL WINAPI **isFlexRayPseudoMsgDynamicType** ()=0
see parameter description of [setFlexRayPseudoMsgProperties\(\)](#)
- virtual void WINAPI **useSatelliteTimeForGPSFormats** (BOOL flag)=0
Set whether to use the satellite time stamp in GPS formats instead of the logger time stamp.
- virtual BOOL WINAPI **isSatelliteTimeForGPSFormats** ()=0
Returns whether to use the satellite time stamp in GPS formats instead of the logger time stamp.
- virtual void WINAPI **setIsochronousMost150Channels** (const char *channels)=0
Set the channel widths of the isochronous channels as comma separated string.
- virtual const char *WINAPI **getIsochronousMost150Channels** ()=0
Returns the channelLabels of the isochronous channels as comma separated string.
- virtual void WINAPI **setAnalogToCANPseudoActive** (BOOL flag)=0
Set whether to activate the analogue data to CAN pseudo message feature.
- virtual void WINAPI **addAnalogPortSettings** (uint16_t analogPort, BOOL isActive, uint32_t canChannel, uint32_t canID, const char *dbcPath)=0
Set analog port settings.
- virtual void WINAPI **clearAnalogPortSettings** ()=0
Clears all port settings set with the [addAnalogPortSettings\(\)](#) function.
- virtual void WINAPI **setDigitalToCANPseudoActive** (BOOL flag)=0
Set whether to activate the digital data to CAN pseudo message feature.
- virtual void WINAPI **addDigitalPortSettings** (uint16_t digitalPort, BOOL isActive, uint32_t canChannel, uint32_t canID, BOOL isExt)=0
Set digital port settings.
- virtual void WINAPI **clearDigitalPortSettings** ()=0
Clears all port settings set with the [addDigitalPortSettings\(\)](#) function.
- virtual void WINAPI **setConvertPhyStatusWithoutData** (BOOL flag)=0
- virtual BOOL WINAPI **isConvertPhyStatusWithoutData** ()=0
- virtual void WINAPI **setUse10GLinkSpeed** (BOOL flag)=0
- virtual BOOL WINAPI **isUse10GLinkSpeed** ()=0

6.8.1 Detailed Description

The [IClientProperties](#) interface replaces the deprecated *ClientProperties* struct.

Call [IBPNGClient::getClientProperties\(\)](#) to get a pointer to an instance of this interface class.

6.8.2 Member Function Documentation

virtual void WINAPI IClientProperties::addAnalogPortSettings (uint16_t *analogPort*, BOOL *isActive*, uint32_t *canChannel*, uint32_t *canID*, const char * *dbcPath*) [pure virtual]

Set analog port settings.

Parameters

<i>analogPort</i>	Analogue port index
<i>isActive</i>	Specifies whether the data of this port should be written to CAN pseudo messages
<i>canChannel</i>	Specifies the CAN channel that should be used for the pseudo messages
<i>canID</i>	Specifies the CAN ID that should be used for the pseudo messages
<i>dbcPath</i>	The path to the DBC file that specifies the signal of the CAN ID's message that should carry the value

virtual void WINAPI IClientProperties::addDigitalPortSettings (uint16_t *digitalPort*, BOOL *isActive*, uint32_t *canChannel*, uint32_t *canID*, BOOL *isExt*) [pure virtual]

Set digital port settings.

Parameters

<i>digitalPort</i>	Digital port index
<i>isActive</i>	Specifies whether the data of this port should be written to CAN pseudo messages
<i>canChannel</i>	Specifies the CAN channel that should be used for the pseudo messages
<i>canID</i>	Specifies the CAN ID that should be used for the pseudo messages
<i>dbcPath</i>	The path to the DBC file that specifies the signal of the CAN ID's message that should carry the value

virtual void WINAPI IClientProperties::setCANPseudoMsgTimeStampProperties (BOOL *writeTimeStampMsg*, uint32_t *channelIndex*, uint32_t *dlc*, uint32_t *canID*, uint32_t *hourBitPos*, uint32_t *minBitPos*, uint32_t *secBitPos*, uint32_t *dayBitPos*, uint32_t *monthBitPos*, uint32_t *yearBitPos*) [pure virtual]

Set CAN pseudo properties for writing time stamp messages.

Parameters

<i>writeTimeS- tampMsg</i>	Active flag for writing periodical CAN pseudo messages with absolute time stamps
<i>channelIndex</i>	CAN channel for the time stamp pseudo messages
<i>dlc</i>	DLC for the time stamp pseudo messages

<i>canID</i>	CAN ID for the time stamp pseudo messages
<i>hourBitPos</i>	Bit position for the hour (0..23, 5 bit length) value in the CAN data bytes
<i>minBitPos</i>	Bit position for the minute (0..59, 6 bit length) value in the CAN data bytes
<i>secBitPos</i>	Bit position for the second (0..59, 6 bit length) value in the CAN data bytes
<i>dayBitPos</i>	Bit position for the day (1..31, 5 bit length) value in the CAN data bytes
<i>monthBitPos</i>	Bit position for the month (1..12, 4 bit length) value in the CAN data bytes
<i>yearBitPos</i>	Bit position for the year (8 bit length) value in the CAN data bytes

virtual void WINAPI IClientProperties::setCANPseudoMsgTriggerProperties (BOOL *writeTriggerMessage*, uint32_t *channelIndex*, uint32_t *dlc*, uint32_t *canID*, uint32_t *triggerNumBitPos*) [pure virtual]

Set CAN pseudo properties for writing trigger messages.

Parameters

<i>writeTriggerMessage</i>	Active flag for writing CAN pseudo messages with trigger information
<i>channelIndex</i>	CAN channel for the trigger pseudo messages
<i>dlc</i>	DLC for the trigger pseudo messages
<i>canID</i>	CAN ID for the trigger pseudo messages
<i>triggerNumBitPos</i>	Bit position for the trigger's index (16 bit length)

virtual void WINAPI IClientProperties::setCommonProperties (const char * *nameOfTester*, int *maxOutputSizeMB*, BOOL *separatedTimeFormat*, BOOL *separatedTimeFormatInOfflineSet*, const char * *alternativeLoggerName*, BOOL *useAlternativeLoggerName*, BOOL *useSubDirectories*, BOOL *midnightSplitting*, BOOL *fileTimeSpansLikeSelection*, BOOL *markerNumberInFileNames*, BOOL *subfolderWithLoggerName*, int *maxOfflineZipSizeMB*, int *maxOutputSizeMBSortedDownload*, BOOL *traceCutterStorage*, const char * *traceCutterFNPatternConversion*, BOOL *createOfflineDataOnTraceCutterStorage*, const char * *traceCutterFNPatternOfflineData*, BOOL *traceCutterMarkerCompact*, BOOL *pauseDataRecordingDuringDownload*, BOOL *tmASCIITimestampPrecisionToMicros*) [pure virtual]

Set Common properties.

Parameters

<i>nameOfTester</i>	Name of tester that is written to the converted file names
<i>maxOutputSizeMB</i>	Maximum file size for converted files. When this size is reached a new file is created.
<i>separatedTimeFormat</i>	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)
<i>separatedTimeFormatInOfflineSet</i>	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)

<i>alternativeLoggerName</i>	The logger device's name is included in the converted files' names. An alternative logger name can be used.
<i>useAlternativeLoggerName</i>	Set this field to 1 if the alternative logger name should be used in converted file names, 0 if not.
<i>useSubDirectories</i>	Set to 1 if converted files should be stored in subdirectories named by their start date, set 0 if they should not.
<i>midnightSplitting</i>	Set to 1 if converted files should be splitted at 00:00:00 of each date, set to 0 if they should not.
<i>fileTimeSpansLikeSelection</i>	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.
<i>markerNumberInFileNames</i>	Specifies whether the indices of the marker included in a converted file should be appended to its file name
<i>subfolderWithLoggerName</i>	Specifies whether the name of the subfolder the converted files are stored in should contain the logger name or not.
<i>maxOutputSizeMBSortedDownload</i>	Maximum file size for sorted download trace files. When this size is reached a new file is created.
<i>tmASCIITimestampPrecisionToMicros</i>	If true the TMASCII on conversion will write 6 digits on timestamp after separator (microsecond resolution)

virtual void WINAPI IClientProperties::setFlexRayPseudoMsgProperties (BOOL *active*, uint8_t *channel*, uint32_t *slotID*, uint32_t *cycleCount*, BOOL *useFlexRayTypeDynamic*)
 [pure virtual]

Set FlexRay pseudo properties.

Parameters

<i>active</i>	Active flag for writing FlexRay pseudo messages for trigger
<i>channel</i>	FlexRayChannel 0 = 1A, 1 = 1B,
<i>slotID</i>	FlexRay slot ID
<i>cycleCount</i>	FlexRay Cycle
<i>useFlexRayTypeDynamic</i>	FlexRay dynamic or static type

virtual void WINAPI IClientProperties::setMOSTPseudoMsgProperties (BOOL *active*, uint32_t *src*, uint32_t *target*, uint32_t *fktBlockID*, uint32_t *fktID*) [pure virtual]

Set MOST pseudo properties.

Parameters

<i>active</i>	Active flag for writing MOST pseudo messages for trigger
<i>src</i>	Source address
<i>target</i>	Target address
<i>fktBlockID</i>	Function block ID
<i>fktID</i>	Function ID

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

- [IClientProperties.h](#)

6.9 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, const char *formatId, int fileId, int offset, int mbnr, bool mappingActive, int mappedChannelId)=0
Adds a channel to the conversion set and assigns the target format to it.
- virtual void [addTimeSpan](#) (uint64_t startTime, uint64_t endTime, uint64_t id=0)=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.
- virtual bool [loadConversionFilters](#) (const char *pathToIni)=0
Conversion filters can be loaded from an ini file.
- virtual bool [loadFormats](#) (const char *pathToIniFile)=0
Loads the format settings from an ini file.
- virtual bool [saveFormats](#) (const char *pathToIniFile) const =0
Saves the format settings to an ini file.

6.9.1 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.

6.9.2 Member Function Documentation

virtual void IConversionSet::addChannel (ChannelType *channelType*, uint8_t *channelIndex*, const char * *formatId*, int *fileId*, int *offset*, int *mbnr*, bool *mappingActive*, int *mappedChannelId*) [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.

Hint for offset, mappingActive and mappedChannelId: Use the configured values! Else the channel will not be found and the data not written. All information can be retrieved from [IChannel](#)

Parameters

<i>channelType</i>	must be one of the appropriate ChannelType enum.
<i>channelIndex</i>	zero-based channel index
<i>formatId</i>	must be one of the appropriate FormatId enum.
<i>fileId</i>	The data of all channels with same formatId and same fileId are written to the same output file. The default value -1 indicates always a separate file for each channel.
<i>offset</i>	Only needed for TSL, default 1. The offset to the original channel number. Can be read out from IChannel
<i>mbnr</i>	Only needed for TSL, default -1. The mainboardnumber of the channels source device. Can be read out from IChannel
<i>mappingActive</i>	Only needed for Channelmapping, default false. If true the mappedChannelId will be used instead of original index. Can be read out from IChannel
<i>mappedChannelId</i>	Only needed for Channelmapping, default -1. If mappingActive is true the mappedChannelId will be used instead of original index. Can be read out from IChannel

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

virtual void IConversionSet::addTimeSpan (uint64_t startTime, uint64_t endTime, uint64_t id = 0) [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)
<i>id</i>	id of timespan, e.g. marker id

virtual bool IConversionSet::loadConversionFilters (const char * pathToIni) [pure virtual]

Conversion filters can be loaded from an ini file.

Currently only some ethernet filters for BLF and pcap-Format are supported. An example ini file can be found in the System Client Libraries package ("conversionFilter.ini").

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

- [BPNGDefines.h](#)

6.10 IFalseMeasureSignal Struct Reference

False measure signal interface.

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

Public Member Functions

- virtual uint8_t [getDeviceId](#) () const =0
Returns the device Id.
- virtual uint16_t [getSignalNo](#) () const =0
Returns the signal number.
- virtual [Reason](#) [getIgnoreReason](#) () const =0
Returns the ignore reason.

6.10.1 Detailed Description

False measure signal interface.

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

- [BPNGDefines.h](#)

6.11 IFalseMeasureSignalList Struct Reference

False measure signal list interface.

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

Public Member Functions

- virtual size_t [getSize](#) () const =0
Returns the number of signals.
- virtual const [IFalseMeasureSignal](#) * [getSignal](#) (size_t index) const =0
Returns the [IFalseMeasureSignal](#) at index.

6.11.1 Detailed Description

False measure signal list interface.

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

- [BPNGDefines.h](#)

6.12 IFormatInfo Struct Reference

FormatInfo interface.

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

Public Member Functions

- virtual const char * [getFormatId](#) () const =0
Returns the FormatId.
- virtual const char * [getName](#) (const char *language) const =0
Returns the format's description name in the language passed as ISO 639-1 language code ("en", "de", etc.)
- 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.
- virtual const char * [getRequiredLicense](#) () const =0
Returns the required license for the format, an empty string for free formats.

6.12.1 Detailed Description

FormatInfo interface.

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

- [BPNGDefines.h](#)

6.13 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.

6.13.1 Detailed Description

Format list interface.

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

- [BPNGDefines.h](#)

6.14 IRdbEvent Struct Reference

Interface to an RDB event.

```
#include <RdbDefines.h>
```

Public Member Functions

- virtual [~IRdbEvent](#) ()
DTOR.
- virtual [RdbEventType](#) WINAPI [getType](#) () const =0
Get type of event.
- virtual uint64_t WINAPI [getUniqueld](#) () const =0
- virtual uint64_t WINAPI [getTimeStamp](#) () const =0
Returns the event's time stamp in usec since 01.01.1970 UTC.
- virtual const char *WINAPI [getTimeZone](#) () const =0
- virtual int WINAPI [getIndex](#) () const =0
Returns the index of this event. Only used for marker events.
- virtual const char *WINAPI [getComment](#) () const =0

6.14.1 Detailed Description

Interface to an RDB event.

6.14.2 Member Function Documentation

virtual const char* WINAPI IRdbEvent::getComment () const [pure virtual]

Returns additional information. The meaning of this string depends on the event's type. See RDB specification document for more information.

virtual const char* WINAPI IRdbEvent::getTimeZone () const [pure virtual]

Returns the logger's time zone that was active at the event's time stamp.

virtual uint64_t WINAPI IRdbEvent::getUniqueld () const [pure virtual]

Returns the unique entry ID that can be set to DataSpans for data download and conversion.
The documentation for this struct was generated from the following file:

- [RdbDefines.h](#)

6.15 IRdbEventList Struct Reference

Interface to a list of rdb events.

```
#include <RdbDefines.h>
```

Public Member Functions

- virtual [~IRdbEventList](#) ()
DTOR.
- virtual size_t WINAPI [getSize](#) () const =0
Returns the size of the event list.
- virtual const [IRdbEvent](#) *WINAPI [getEvent](#) (size_t index) const =0
Returns a pointer to the [IRdbEvent](#) at index.

6.15.1 Detailed Description

Interface to a list of rdb events.

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

- [RdbDefines.h](#)

6.16 IRdbTraceBlock Struct Reference

Public Member Functions

- virtual [~IRdbTraceBlock](#) ()
DTOR.
- virtual uint64_t WINAPI **getUniqueld** () const =0
- virtual uint64_t WINAPI **getStartTimeStamp** () const =0
- virtual uint64_t WINAPI **getEndTimeStamp** () const =0
- virtual const char *WINAPI **getTimeZone** () const =0
- virtual const char *WINAPI **getLoggerModuleName** () const =0
- virtual const char *WINAPI **getFilePath** () const =0
- virtual const char *WINAPI **getFileName** () const =0
- virtual uint64_t WINAPI **getDataFileSize** () const =0
- virtual uint64_t WINAPI **getDataSize** () const =0
- virtual uint64_t WINAPI **getBlockNumber** () const =0
- virtual const char *WINAPI **getCfgBackupFile** () const =0
- virtual const char *WINAPI **getDataColumnValue** (const char *columnName) const =0
- virtual const char *WINAPI **getComment** () const =0

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

- [RdbDefines.h](#)

6.17 IRdbTraceBlockList Struct Reference

Public Member Functions

- virtual [~IRdbTraceBlockList](#) ()
DTOR.
- virtual size_t WINAPI [getSize](#) () const =0
Returns the size of the event list.
- virtual const [IRdbTraceBlock](#) *WINAPI [getTraceBlock](#) (size_t index) const =0
Returns a pointer to the [IRdbEvent](#) at index.

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

- [RdbDefines.h](#)

6.18 ITesttoolsChannel Struct Reference

Channel interface.

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

Public Member Functions

- virtual [IChannel](#) * [getIChannel](#) () const =0
Returns the [IChannel](#) of this [ITesttoolsChannel](#).
- virtual BOOL [matchIChannel](#) (const [IChannel](#) *iChannel) const =0
Returns whether the channel matches with the contained channel.
- virtual uint32_t [getContainerId](#) () const =0
Returns the channel's containerId.
- virtual uint32_t [getPseudoContainerId](#) () const =0
Returns the channel's associated containerId.
- virtual const char * [getPseudoChannelName](#) () const =0
Returns the channel's associated containerId name.
- virtual uint16_t [getBaseCanId](#) () const =0
Returns the channel's containerId.
- virtual bool [isExtendedCanId](#) () const =0
Returns the channel's containerId is extended or not.
- virtual const char * [getHostIp](#) () const =0
Returns the ethernet host ip.
- virtual const char * [getDeviceIp](#) () const =0
Returns the ethernet device ip.
- virtual unsigned int [getDevicePort](#) () const =0
Returns the ethernet device ip.
- virtual int [getProtocol](#) () const =0
Returns protocol.
- virtual int [getDebugLevel](#) () const =0
Returns debuglevel.

6.18.1 Detailed Description

Channel interface.

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

- [BPNGDefines.h](#)

6.19 ITesttoolsChannelList Struct Reference

TesttoolsChannel list interface.

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

Public Member Functions

- virtual int [getSize](#) () const =0
Returns the number of channels.
- virtual const [ITesttoolsChannel](#) * [getTesttoolsChannel](#) (int index) const =0
Returns the [ITesttoolsChannel](#) at index.

6.19.1 Detailed Description

TesttoolsChannel list interface.

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

- [BPNGDefines.h](#)

6.20 LoginData Struct Reference

structure for login

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

Public Attributes

- const char * **userName**
- const char * **userPwd**

6.20.1 Detailed Description

structure for login

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

- [BPNGDefines.h](#)

6.21 MemoryFillLevel Struct Reference

stores memory fill level of a device

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

Public Attributes

- uint32_t [ringBufferSize](#)
size of ringbuffer in GB
- uint8_t [percentageFill](#)
percentage filled
- uint8_t [percentageFillProtected](#)
percentage filled of protected areas
- uint32_t [extRingBufferSize](#)
size of external media ringbuffer in GB
- uint8_t [extPercentageFill](#)
external media percentage filled
- uint8_t [extPercentageFillProtected](#)
external media percentage filled of protected areas
- uint64_t [mbnr](#)
mainboardnumber of device
- uint32_t [secondRingBufferSize](#)
size of the second ringbuffer in GB (only available on bluePiraT Rapid)
- uint8_t [secondPercentageFill](#)
second ringbuffer percentage filled
- uint8_t [secondPercentageFillProtected](#)
second ringbuffer percentage filled of protected areas

6.21.1 Detailed Description

stores memory fill level of a device

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

- [BPNGDefines.h](#)

6.22 OnlineLoggerInfo Struct Reference

Struct with information about a logger found in LAN/WLAN used to notify [IBPNGClientListener](#) about detected/disappeared devices.

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

Public Attributes

- const char * [ip](#)
the logger's ip address, obligatory if [OnlineLoggerInfo](#) is used with [IBPNGClient::setDevice\(\)](#)
- const char * [name](#)
the logger's name
- const char * [mbnr](#)
mainboard number
- const char * [deviceSN](#)
device serial number, since FW 2.2.1
- uint8_t [occupied](#)
0 = not occupied, 1 = connected with client, 2 = occupied by temp config (via external media)
- const char * [currentUser](#)
user name of connected pc account
- uint8_t [loggerStatus](#)
current logger status,
- uint8_t [wlan](#)
Flag for connection type. 0 = ethernet, 1 = wlan.
- const char * [tslEth0IP](#)
ip address of device connected to eth0, 0.0.0.0 if none
- const char * [tslEth1IP](#)
ip address of device connected to eth1, 0.0.0.0 if none
- int8_t [tslId](#)
id for device in tsl network, continues in tsl, starts with 0 on first device
- int32_t [tslNetworkId](#)
id of tsl network, -1 = no TSL, all devices with same tslNetworkId belong to the same TSL
- const char * [tslName](#)
name(id) of tsl network
- uint8_t [deviceType](#)
Device type,.
- const char * [fwVersion](#)
Current firmware version, since FW 2.1.1.
- uint16_t [tmpBusPort](#)
tmp bus port
- uint16_t [udpPort](#)

- udp port for keep alive*
- uint16_t [ftpPort](#)
ftp port
- uint8_t [isNotResponding](#)
device responding status
- const char * [sfplp](#)
logic IP address for SFP transfer
- uint16_t [sfpPort](#)
SFP server port.

6.22.1 Detailed Description

Struct with information about a logger found in LAN/WLAN used to notify [IBPNGClientListener](#) about detected/disappeared devices.

If you want to connect to a device by setting a representation of it via `IBPNGClient::setDevice(OnlineLoggerInfo device)` followed by a call of [IBPNGClient::connect\(\)](#) only the IP parameter is obligatory.

Example:

```
IBPNGClient* client = getBPNGClient();
OnlineLoggerInfo device = createEmptyOnlineLoggerInfo();
device.ip = "192.168.0.233";
client->setDevice(device);
client->connect();
```

See also

[IBPNGClient::scanNetworkForLogger\(\)](#), [IBPNGClientListener](#)

6.22.2 Member Data Documentation

uint8_t OnlineLoggerInfo::deviceType

Device type,.

See also

[BPNGDeviceType](#)

uint8_t OnlineLoggerInfo::loggerStatus

current logger status,

See also

[BPNGLoggerStatus](#)

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

- [BPNGDefines.h](#)

6.23 OnlineLoggerInfoStringPair Struct Reference

a helper object for configuration, license update or firmwareupdate: a key value pair for assigning a configuration, licensefile, etc. to a device

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

Public Attributes

- [OnlineLoggerInfo](#) **key**
the device
- `const char *` **value**
the value, for example a path to a firmware update packet

6.23.1 Detailed Description

a helper object for configuration, license update or firmwareupdate: a key value pair for assigning a configuration, licensefile, etc. to a device

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

- [BPNGDefines.h](#)

6.24 RdbEvent2 Struct Reference

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

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

Public Member Functions

- **RdbEvent2** (const [IRdbEvent](#) *rdbEvent)

Public Attributes

- [RdbEventType](#) **type**
- `uint64_t` **uniqueID**
- `uint64_t` **timeStamp**
- `std::string` **timeZone**
- `int` **index**
- `std::string` **comment**

6.24.1 Detailed Description

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

To achieve a compiler independend interface for the Telemotive 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:

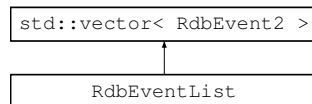
- [RdbEventList.hh](#)

6.25 RdbEventList Class Reference

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

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

Inheritance diagram for RdbEventList:



Public Member Functions

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

6.25.1 Detailed Description

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

To achieve a compiler independent interface for the Telemotive 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](#)

6.26 TSLCluster Struct Reference

Representation of a chain of Telemotive devices combined via Telemotive System Link (TSL)

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

Public Attributes

- uint8_t **numDevices**
- [OnlineLoggerInfo](#) * **loggerArray**

6.26.1 Detailed Description

Representation of a chain of Telemotive devices combined via Telemotive System Link (TSL)

An instance of this struct must be used if [IBPNGClient](#) should connect to a [TSLCluster](#). @sa [IBPNGClient::setTSLCluster\(\)](#)

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

- [BPNGDefines.h](#)

Chapter 7

File Documentation

7.1 BPNGDefines.h File Reference

Defines for Telemotive Client Library.

```
#include "cstdio"
#include "stdint.h"
```

Classes

- struct [IFalseMeasureSignal](#)
False measure signal interface.
- struct [IFalseMeasureSignalList](#)
False measure signal list interface.
- struct [IChannel](#)
Channel interface.
- struct [ITesttoolsChannel](#)
Channel interface.
- struct [IChannelList](#)
Channel list interface.
- struct [ITesttoolsChannelList](#)
TesttoolsChannel list interface.
- struct [IFormatInfo](#)
FormatInfo interface.
- struct [IFormatList](#)
Format list interface.
- struct [IConversionSet](#)
A conversion set stores all conversion relevant settings.
- struct [OnlineLoggerInfo](#)
Struct with information about a logger found in LAN/WLAN used to notify [IBPNGClientListener](#) about detected/disappeared devices.
- struct [TSLCluster](#)
Representation of a chain of Telemotive devices combined via Telemotive System Link (TSL)
- struct [DataSpan](#)
- struct [BPNGError](#)

Error struct with error code and optional error message.

- struct [LoginData](#)
structure for login
- struct [MemoryFillLevel](#)
stores memory fill level of a device
- struct [OnlineLoggerInfoStringPair](#)
a helper object for configuration, license update or firmwareupdate: a key value pair for assigning a configuration, licensefile, etc. to a device

Macros

- #define **BPNGDEFINES_H**
- #define **WINAPI**
- #define **DECLDIR**
- #define **BOOL** bool
- #define **VOID** void
- #define **CLIENT_LIB_VERSION** "5.0.4"

Typedefs

- 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_AMBIGUOUS](#) = 66,
[BPNG_FAILED_TO_CONNECT_STREAMING](#) = 67,
[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](#) = 28,
[BPNG_INITIALISATION_ERROR](#) = 29,
[BPNG_RDB_SQLITE_QUERY_ERROR](#) = 30, [BPNG_RDB_OPEN_FAILED](#) = 31, [BPNG_CONVERSION_ERROR](#) = 32,
[BPNG_CONV_SET_NOT_FOUND](#) = 33,
[BPNG_NOTHING_TO_CONVERT](#) = 34, [BPNG_TMT_FILE_ID_ERROR](#) = 35, [BPNG_TMT_FORMAT_ERROR](#) = 36,
[BPNG_TMT_FORMAT_ERROR_TS](#) = 37,
[BPNG_INVALID_MESSAGE_ERROR](#) = 38, [BPNG_INVALID_MESSAGE_ID](#) = 39, [BPNG_INVALID_MESSAGE](#) = 40,
[BPNG_INVALID_MESSAGE_SUBID](#) = 41,
[BPNG_INVALID_MESSAGE_LEN](#) = 42, [BPNG_CONV_FORMAT_ERROR](#) = 43, [BPNG_DOWNLOAD_ERROR](#) = 44,

```

= 44, BPNG_NOTHING_TO_DOWNLOAD = 45,
BPNG_INVALID_OFFLINE_SET = 46, BPNG_PARAMETER_MISMATCH = 47, BPNG_FW_VERSION_CHECK = 48,
BPNG_USER_CANCELLED = 49,
BPNG_MIN_VERSION_ERROR = 50, BPNG_EXCEPTION = 51, BPNG_INCOMPATIBLE_RDB = 52,
BPNG_UNSPECIFIED_ERROR = 53,
BPNG_LOAD_DBC_FAILED = 81, 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, BPNG_INDEX_OUT_OF_RANGE = 59,
BPNG_READ_CONFIG_BACKUP_ERR = 60,
BPNG_INVALID_RPC_COMMAND = 61, BPNG_INVALID_TSL_CASCDING = 62, BPNG_LOGIN_CANCELLED = 63,
BPNG_USER_PWD_WRONG = 64,
BPNG_NO_ACCESS_FOR_FUNCTION = 65, BPNG_STREAMING_PROTOCOL_ERROR = 68,
BPNG_STREAMING_SOCKET_ERROR = 69, BPNG_STREAMING_DISABLED = 70,
BPNG_FW_DEPRECATED = 71, BPNG_STREAMING_ABORTED_BY_PEER = 72, BPNG_INCONSISTENT_TSL_CLUSTER = 80,
BPNG_INVALID_TSL_CLUSTER = 82,
BPNG_DLL_NO_FORMAT_PLUGIN = 83, BPNG_FORMAT_PLUGIN_ID_EXISTS = 84,
BPNG_DLL_NO_SYSTEMCLIENTLISTENER_PLUGIN = 90, BPNG_FAILED_RENAME_RESUMED_OFFLINE = 85,
BPNG_FAILED_RENAME_RESUMED_RDB = 86, BPNG_RESUME_INIT_FAILURE = 87,
BPNG_SIGNAL_FILTER_INVALID_CONFIG = 88, BPNG_BAD_ALLOC = 89,
BPNG_INVALID_FN_PATTERN = 91, BPNG_NOTHING_TO_TEST_REPORT = 92, BPNG_NO_TRACE_DATA = 93,
BPNG_FUNCTION_NOT_AVAILABLE = 94,
BPNG_SFP_FILE_TRANSFER_NOT_ACKNOWLEDGED = 95, BPNG_SFP_LOCAL_ERROR = 96,
BPNG_SFP_NO_DATA_RECEIVED = 97, BPNG_MISSING_SFP_IP = 98,
BPNG_TMT_CHECKSUM_ERROR = 99, BPNG_PBFUPDATE_FAILED = 100, BPNG_EXCEPTION_MINI = 101 }

```

enum Error codes

- enum FWUpdateErrorCode {
 FWUPDATE_ERRORCODE_NO_ERR = 0, FWUPDATE_ERRORCODE_FW_PKT_NAME_EMPTY = -2,
 FWUPDATE_ERRORCODE_FW_PKT_MISSING = -3, FWUPDATE_ERRORCODE_NAMED_PIPE_SERVER_OPEN = -4,
 FWUPDATE_ERRORCODE_NAMED_PIPE_SERVER_OPEN = -5, FWUPDATE_ERRORCODE_FW_UPDATE_FAILED = -6,
 FWUPDATE_ERRORCODE_MISSING_LINUX_DISTRO = -7, FWUPDATE_ERRORCODE_MISSING_LINUX_KERNEL = -8,
 FWUPDATE_ERRORCODE_MISSING_TMLIB_FILE = -9, FWUPDATE_ERRORCODE_MISSING_ATOM_FIRMWARE = -10,
 FWUPDATE_ERRORCODE_MISSING_CLIENT_FILE = -11, FWUPDATE_ERRORCODE_MISSING_FPGAB_FILE = -12,
 FWUPDATE_ERRORCODE_MISSING_FPGAB_FILE = -13, FWUPDATE_ERRORCODE_MISSING_EXTENSION_BOARD_VARIANCE = -14,
 FWUPDATE_ERRORCODE_MISSING_GBE_FILE = -15, FWUPDATE_ERRORCODE_MISSING_SBC_FLASH_SCRIPT = -16,
 FWUPDATE_ERRORCODE_MISSING_SBC_FLASH_SCRIPT = -17, FWUPDATE_ERRORCODE_MISSING_RCV_FILE = -18,
 FWUPDATE_ERRORCODE_MISSING_RCV_FILE = -19, FWUPDATE_ERRORCODE_MISSING_LINUX_KERNEL = -20,
 FWUPDATE_ERRORCODE_UNKNOWN_MB_HW_VERSION = -21, FWUPDATE_ERRORCODE_UNKNOWN_EXTENSION_BOARD_VARIANCE = -22,
 FWUPDATE_ERRORCODE_UNKNOWN_EXTENSION_BOARD_VARIANCE = -23,
 FWUPDATE_ERRORCODE_NOT_READABLE_EXTENSION_BOARD_VARIANCE = -24,
 FWUPDATE_ERRORCODE_NOT_READABLE_EXTENSION_BOARD_HW_VERSION = -25,
 FWUPDATE_ERRORCODE_NOT_READABLE_HW_TYPE_VERSION = -26, FWUPDATE_ERRORCODE_FAILED_UPDATE_APP_LIBS = -27,
 FWUPDATE_ERRORCODE_FAILED_UPDATE_APP_LIBS = -28,
 FWUPDATE_ERRORCODE_FAILED_UPDATE_GBEC = -29, FWUPDATE_ERRORCODE_CONV_CFG_ERROR = -30,
 FWUPDATE_ERRORCODE_FAILED_UNCOMPRESS_LINUX_KERNEL = -31, FWUP-

```

DATE_ERRORCODE_FAILED_UNCOMPRESS_LINUX_KERNEL_MODULES = -32,
FWUPDATE_ERRORCODE_FAILED_CPY_LINUX_KERNEL = -33, FWUPDATE_ERRORCODE_FAILED_U
= -34, FWUPDATE_ERRORCODE_FAILED_CPY_CLIENT_FILE = -35, FWUPDATE_ERRORCODE_FAILED
= -36,
FWUPDATE_ERRORCODE_FAILED_SBC_FLASH = -37, FWUPDATE_ERRORCODE_FAILED_UPDATE_C
= -38, FWUPDATE_ERRORCODE_FAILED_UPDATE_CCP_XCP_SEED_KEY_SERVERS
= -39, FWUPDATE_ERRORCODE_MISSING_CCP_XCP_FILE = -40,
FWUPDATE_ERRORCODE_MISSING_CCP_XCP_SEED_KEY_SERVER_FILE = -41, FWUP-
DATE_ERRORCODE_MISSING_SPYNIC_FILE = -42, FWUPDATE_ERRORCODE_MISSING_LOADING_IS
= -43, FWUPDATE_ERRORCODE_MISSING_DEVICE_FPGAB_FILE = -44,
FWUPDATE_ERRORCODE_MISSING_DEVICE_FPGAA_FILE = -45, FWUPDATE_ERRORCODE_UNREAL
= -46, FWUPDATE_ERRORCODE_LINUX_KERNEL_MAY_FREEZE_SYSTEM = -47, FWUP-
DATE_ERRORCODE_NOT_SET_DEVICE_PATH = -48,
FWUPDATE_ERRORCODE_NOT_SET_FW_FILE = -49, FWUPDATE_ERRORCODE_NOT_SET_FPGA_K
= -50, FWUPDATE_ERRORCODE_MISSING_DEVICE_FILE = -51, FWUPDATE_ERRORCODE_MISSING_F
= -52,
FWUPDATE_ERRORCODE_MISSING_TMUDEVQ = -53, FWUPDATE_ERRORCODE_UNKNOWN_DEVICE
= -54, FWUPDATE_ERRORCODE_FAILED_CPY_FPGA_FILE = -55, FWUPDATE_ERRORCODE_FAILED
= -56,
FWUPDATE_ERRORCODE_ERROR_FLASH_FPGA = -57, FWUPDATE_ERRORCODE_ERROR_FLASH_S
= -58, FWUPDATE_ERRORCODE_ERROR_LOADING_FPGA_JTAG_DRIVER = -59, FWUP-
DATE_ERRORCODE_MISSING_EXTENSION_BOARD_VIA_PCIE = -60,
FWUPDATE_ERRORCODE_FAILED_CONV_CFG = -61, FWUPDATE_ERRORCODE_FAILED_UPLOAD
= -62, FWUPDATE_ERRORCODE_FWUFOLDER_EXISTS = 63, FWUPDATE_ERRORCODE_UNDEFINED
= -1 }
• enum BPNGWarningCode {
    BPNG_NOWARNING, BPNG_WARNING_CLOSE_TRACE_FILES, BPNG_WARNING_MESSAGES_NOT_C
    BPNG_WARNING_NO_ESO_TRACE,
    BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONES, BPNG_WARNING_RECOVERING_FAILED
}
    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_CLIENT
    = 3,
    BR_FULL_ALL_TRACES = 4, BR_FULL_TIMESPAN_TRACES = 5 }
    Mode for the IBPNG::downloadBugReport() function.
• enum ChannelType {
    CH_UNDEFINED = 0, OBSOLETE_CH_CANLS, CH_CAN, 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_MOST150_STREAM, CH_ANALOG_IN, CH_DIGITAL_IN,
    CH_CAMERA, CH_CCPXCP, CH_DIAG, CH_GPS,
    CH_ECL, CH_COMPLEXFILTER, CH_TTY, CH_MII }
    Currently supported interfaces.
• enum PwdPrivilegesFuncId {
    REMOVE_DATA = 0, SET_TIME, SET_EVENT, RECONFIG,
    RECONFIG_PASSWORD, RECONFIG_COMPLEX_FILTER, UPLOAD_WINE_DLLS, UP-
    DATE_FIRMWARE,
    CHANGE_LICENCES, PRIVILEGES_END }
• enum Reason {
    R_UNSUPPORTED_BIT_MASK, R_BIT_MASK_OVERLAP, R_UNSUPPORTED_COMPU_TAB,

```

- `R_FORBIDDEN_TAB_VALUE,`
`R_UNKNOWN }`
- enum `BPNGLoggerStatus` {
`LS_OK = 0, LS_ERROR = 1, LS_NOSYNC = 2, LS_WARNING = 3,`
`LS_FWUPDATE = 4, LS_MEM = 5, LS_RING = 6, LS_UNDEFINED = -1 }`
Logger status.
- enum `BPNGDeviceType` {
`DEV_BP2, DEV_BPMINI, DEV_BP2_V1X, DEV_BP2_V2X,`
`DEV_RC_TOUCH, DEV_BP_REMOTE, DEV_BP_TOUCH, DEV_RAPID,`
`DEV_TRACE_COLL, DEV_TRACE_COLL_DS, DEV_TSL = 0x80, DEV_UNKNOWN = 0xFF`
`}`
Enumartion of Telemotives next generation data loggers.
- enum `DataSpanType` { `DST_IDSPAN = 0, DST_TIMESPAN = 1 }`
Types for `DataSpan`.

7.1.1 Detailed Description

Defines for Telemotive Client Library.

Author

Markus van Pinxteren

Date

12.05.2010

7.1.2 Enumeration Type Documentation

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.

enum BPNGDeviceType

Enumartion of Telemotives next generation data loggers.

Enumerator

- `DEV_BP2`** **Deprecated** For blue PiraT 2 devices use type `DEV_BP2_V1X`, for new blue PiraT 2 5E devices use `DEV_BP2_V2X`
- `DEV_BPMINI`** blue PiraT mini devices
- `DEV_BP2_V1X`** standard blue PiraT 2 device

DEV_BP2_V2X blue PiraT 2 5E device
DEV_RC_TOUCH Remote Control Touch.
DEV_BP_REMOTE blue PiraT Remote
DEV_BP_TOUCH blue PiraT Touch
DEV_RAPID blue PiraT Rapid
DEV_TRACE_COLL Trace Collector.
DEV_TRACE_COLL_DS Trace Collector Download Station.
DEV_TSL internal use only! don't use!

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_AMBIGUOUS_IP multiple devices with same IP available
BPNG_FAILED_TO_CONNECT_STREAMING Streaming feature could not be 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::initOffline` 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 thus 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.

BPNG_INVALID_RPC_COMMAND if a rpc command for tsl is wrong

BPNG_INVALID_TSL_CASCDING if cascading of tsl is invalid

BPNG_INCONSISTENT_TSL_FWVERSIONS if fw versions on tsl clusters are inconsistent

BPNG_INVALID_TSL_CLUSTER in case of different TSLNetwork IDs

BPNG_NOTHING_TO_TEST_REPORT There are no test drive data spans available that could be converted.

BPNG_NO_TRACE_DATA_ON_DOWNLOAD_STATION No trace data found on download station.

BPNG_EXCEPTION_MINI_DUMP Some kind of unhandled exception was thrown. MiniDump file was written.

enum BPNGLoggerStatus

Logger status.

Enumerator

LS_OK Device is ok.

LS_ERROR Device has at least one active error.

LS_NOSYNC Device is configured as slave but no master is found.

LS_WARNING Device has at least one active warning.

LS_FWUPDATE Firmware update in progress.

LS_MEM Internal storage of device is full. Ring buffer deactivated or full with protected trace files.

LS_RING Internal storage of device is full. Ring buffer is activated.

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.

BPNG_WARNING_NO_ESO_TRACE ethernet data for eso trace conversion is not logged in eso trace format

BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONES A TSL cluster with loggers with different time zones is in undefined state. It's not defined which time zone will be used for time zone dependent processes.

BPNG_WARNING_RECOVERING_FAILED Recovering trace files from a previous power down failed.

enum ChannelType

Currently supported interfaces.

Enumerator

CH_UNDEFINED undefined channel type
OBSOLETE_CH_CANLS CAN low speed interface.
CH_CAN 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_MOST150_STREAM MOST 150 synchronous channel (streaming data)
CH_ANALOG_IN Analog in.
CH_DIGITAL_IN Digital in.
CH_CAMERA Camera channel.
CH_CCPXCP CCP XCP.
CH_DIAG Diagnose, currently not used.
CH_GPS Global Positioning System.
CH_ECL Electronic Control Line.
CH_TTY TTY channel for QXDM.
CH_MII MII channel for ethernet spy.

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

enum Reason

Enumerator

R_UNSUPPORTED_BIT_MASK DBC file don't support bit operations with a bit mask.

R_BIT_MASK_OVERLAP Bit mask is incorrect and cause a overlap with at least one other signal.

R_UNSUPPORTED_COMPU_TAB DBC file don't support all compu tab types; only tab will ignored, not the signal itself!

R_FORBIDDEN_TAB_VALUE DBC file don't support all possible values of a compu tab; only tab will ignored, not the signal itself!

R_UNKNOWN Unknown reason.

7.2 BPNGLoggerDetector.hh File Reference

Logger Detector Sample.

```
#include "IBPNGClient.h"
#include "IBPNGClientListener.h"
#include "TSLClusterImpl.hh"
#include <string>
#include <sstream>
```

Classes

- class [BPNGLoggerDetector](#)

7.2.1 Detailed Description

Logger Detector Sample.

7.3 IBPNGClient.h File Reference

Interface class for the BPNGClient DLL.

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

Classes

- struct [IBPNGClient](#)
Interface class for the Telemotive Client Library.

Functions

- DECLDIR const char *WINAPI [getLibVersion](#) ()
Returns the current client library version.
- DECLDIR [IBPNGClient](#) *WINAPI [getBPNGClient](#) (const char *name="")

Factory function that creates instances of BPNGClient giving away ownership.

- DECLDIR [BPNGErrCode](#) WINAPI [getNumTSLMemberFromOfflineDataSet](#) (const char *offlinePath, int *numMember)
- DECLDIR void WINAPI [setTempDir](#) (const char *tmp)

Sets the directory where all temporary files are created. If not called, the default system's tmp dir is used.
- DECLDIR const char *WINAPI [getTempDir](#) ()
- DECLDIR void WINAPI [setLanguageID](#) ([LanguageID](#) id)

Sets the language for status messages.
- DECLDIR [IConversionSet](#) *WINAPI [createNewConversionSet](#) ()

returns a new created conversionset
- DECLDIR void WINAPI [freeConversionSetMemory](#) ([IConversionSet](#) *convSet)
- DECLDIR [IClientProperties](#) *WINAPI [createNewClientProperties](#) ()
- DECLDIR void WINAPI [freeClientPropertiesMemory](#) ([IClientProperties](#) *prop)
- DECLDIR void WINAPI [writeLogFile](#) (const char *path, int maxSizeInByte, int numBackupFiles)
- DECLDIR void WINAPI [writeLogToCout](#) (bool flag)
- DECLDIR void WINAPI [writeLogToDebugView](#) (bool flag)
- 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.
- DECLDIR [OnlineLoggerInfo](#) [createEmptyOnlineLoggerInfo](#) ()

7.3.1 Detailed Description

Interface class for the BPNGClient DLL.

Author

Markus van Pinxteren

Date

21.04.2010

7.3.2 Function Documentation

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

Adds a log listener to the library.

If you want to receive the debug outputs from the client library, 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](#)

DECLDIR OnlineLoggerInfo createEmptyOnlineLoggerInfo ()

Creates an empty [OnlineLoggerInfo](#). Use this function for devices if the LoggerDetector won't work in your network.

Don't forget to fill the necessary fields.

See also

[OnlineLoggerInfo](#)

DECLDIR IClientProperties* WINAPI createNewClientProperties ()

After modifying the properties, you can set them to an instance of [IBPNGClient](#) with `setClientProperties()`;

See also

[IClientProperties](#), `setClientProperties()`

DECLDIR void WINAPI freeClientPropertiesMemory (IClientProperties * *prop*)

To free memory of [IClientProperties](#), use this method. Otherwise the memory will be freed when detaching the DLL from process. Never call any function of an [IClientProperties](#) pointer after passing the pointer to the `freeClientPropertiesMemory` function. This would cause a heap corruptions.

DECLDIR void WINAPI freeConversionSetMemory (IConversionSet * *convSet*)

To free memory of a [IConversionSet](#), use this method. Otherwise the memory will be freed when detaching the DLL from process. Never call any function of an [IConversionSet](#) after passing the pointer to the `freeConversionSetMemory` function. This would cause a heap corruptions.

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

Factory function that creates instances of [BPNGClient](#) giving 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\(\)](#)

DECLDIR BPNGErrCode WINAPI getNumTSLMemberFromOfflineDataSet (const char * *offlinePath*, int * *numMember*)

Read out the number of TSL members from a offline data set.

DECLDIR void WINAPI writeLogFile (const char * *path*, int *maxSizeInByte*, int *numBackupFiles*)

From version 2.1.1 on the client library doesn't write log messages to `std::cout` by default. The lib actually doesn't write a log at all unless this function is called. A log file is created under the passed *path*. If the file already exists, the logs will be appended. The file will be closed when the DLL is detached from the process.

DECLDIR void WINAPI writeLogToCout (bool *flag*)

The library's log output can also be written to `std::cout`. If this is required activate cout log with this function. Default is no cout output.

7.4 IBPNGClientListener.h File Reference

Interface class for the BPNGClient listener.

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

Classes

- struct [IBPNGClientListener](#)

7.4.1 Detailed Description

Interface class for the BPNGClient listener.

Author

Markus van Pinxteren

Date

12.05.2010

7.5 IClientProperties.h File Reference

Interface for client properties.

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

Classes

- struct [IClientProperties](#)

*The [IClientProperties](#) interface replaces the deprecated *ClientProperties* struct.*

7.5.1 Detailed Description

Interface for client properties.

Author

Markus van Pinxteren

Date

20.03.2014

7.6 RdbDefines.h File Reference

Public interfaces for Telemotive Reference Database access.

```
#include <atom-config.h>
#include <cstdlib>
#include <stdint.h>
```

Classes

- struct [IRdbEvent](#)
Interface to an RDB event.
- struct [IRdbEventList](#)
Interface to a list of rdb events.
- struct [IRdbTraceBlock](#)
- struct [IRdbTraceBlockList](#)

Enumerations

- enum [RdbEventType](#) {
UNKNOWN = 0, **STARTUP** = 0x01, **SHUTDOWN** = 0x02, **MARKER** = 0x03,
INFO = 0x05, **SLAVE_OFFSET** = 0x06, **SLAVE_TO_MASTER** = 0x07, **DATA_DELETED** =
0x08,
TIME_SET = 0x09, **NEW_TIME** = 0x0A, **SUDDEN_DEATH** = 0x0B, **TSL_SLAVE_OFFSET**
= 0x0C,
TSL_SLAVE_TO_MASTER = 0x0D, **TSL_SESSION_START** = 0x0E, **TSL_SESSION_END**
= 0x0F, **CONFIG** = 0x10,
WAKEUP = 0x11, **START_TESTDRIVE** = 0x12, **STOP_TESTDRIVE** = 0x13, **TESTDRIVE_INFO**
= 0x14 }

7.6.1 Detailed Description

Public interfaces for Telemotive Reference Database access.

7.6.2 Enumeration Type Documentation

enum RdbEventType

See also

tmlib's eventID.hh

Enumerator

STARTUP bp2 startup
SHUTDOWN bp2 shutdown
MARKER Marker set.
INFO Info is set.
SLAVE_OFFSET cascading slave offset
SLAVE_TO_MASTER cascading slave to master
DATA_DELETED All data and data space is deleted.
TIME_SET bp2 time was set

NEW_TIME new time
SUDDEN_DEATH no "real" shutdown was found after startup.
TSL_SLAVE_OFFSET slave is synced with master.
TSL_SLAVE_TO_MASTER slave is not synced with master.
TSL_SESSION_START start of a tsl synchronized session
TSL_SESSION_END end of a tsl synchronized session
CONFIG configuration has been updated
WAKEUP bpng wake-up source
START_TESTDRIVE start an easy track test drive
STOP_TESTDRIVE stop an easy track test drive
TESTDRIVE_INFO misc. info event for test drive data like testname, vin, map-version, reproducibility, etc.

7.7 RdbEventList.hh File Reference

[IRdbEvent](#) wrapper.

```

#include <vector>
#include <string>
#include "BPNGDefines.h"

```

Classes

- struct [RdbEvent2](#)
Implementation class for a wrapper of [IRdbEvent](#) using STL classes.
- class [RdbEventList](#)
Implementation class for a wrapper of [IRdbEventList](#) using STL classes.

7.7.1 Detailed Description

[IRdbEvent](#) wrapper.

Index

- activateGatewayLoggerDetection
 - IBPNGClient, [36](#)
- addAnalogPortSettings
 - IClientProperties, [63](#)
- addChannel
 - IConversionSet, [67](#)
- addDigitalPortSettings
 - IClientProperties, [64](#)
- addLogListener
 - IBPNGClient.h, [89](#)
- addRdbldRange
 - IConversionSet, [68](#)
- addTimeSpan
 - IConversionSet, [68](#)
- assignDBCFile
 - IBPNGClient, [37](#)
- assignDatabaseFile
 - IBPNGClient, [36](#)
- BPNG_AMBIGUOUS_IP
 - BPNGDefines.h, [84](#)
- BPNG_CONNECT_FTP_FAILED
 - BPNGDefines.h, [84](#)
- BPNG_CONNECT_TMPBUS_FAILED
 - BPNGDefines.h, [84](#)
- BPNG_CONV_FORMAT_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_CONV_SET_NOT_FOUND
 - BPNGDefines.h, [85](#)
- BPNG_CONVERSION_ERRORS
 - BPNGDefines.h, [85](#)
- BPNG_DIR_EXISTS_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_DOWNLOAD_ERRORS
 - BPNGDefines.h, [85](#)
- BPNG_ENGLISH
 - BPNGDefines.h, [87](#)
- BPNG_EXCEPTION
 - BPNGDefines.h, [86](#)
- BPNG_EXCEPTION_MINI_DUMP
 - BPNGDefines.h, [86](#)
- BPNG_FAILED_TO_CONNECT_STREAMING
 - BPNGDefines.h, [84](#)
- BPNG_FAILED_TO_CREATE_LOCAL_FILE_OR_DIRECTORY
 - BPNGDefines.h, [85](#)
- BPNG_FILE_EXISTS_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_FTP_CHANGE_CWD_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_CREATE_REMOTE_DIR_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_LOGIN_FAILED
 - BPNGDefines.h, [84](#)
- BPNG_FTP_NOT_CONNECTED
 - BPNGDefines.h, [84](#)
- BPNG_FTP_READ_REMOTE_FILE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_REMOTE_PATH_NOT_FOUND
 - BPNGDefines.h, [84](#)
- BPNG_FTP_REMOVE_REMOTE_DIR_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_REMOVE_REMOTE_FILE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_SERVER_NOT_FOUND
 - BPNGDefines.h, [84](#)
- BPNG_FTP_TRANSFER_USER_CANCELED
 - BPNGDefines.h, [84](#)
- BPNG_FTP_WRITE_REMOTE_FILE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FW_VERSION_CHECK_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_GERMAN
 - BPNGDefines.h, [87](#)
- BPNG_INCOMPATIBLE_RDB
 - BPNGDefines.h, [86](#)
- BPNG_INCONSISTENT_TSL_FWVERSIONS
 - BPNGDefines.h, [86](#)
- BPNG_INITIALISATION_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_MESSAGE_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_MESSAGE_ID
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_MESSAGE_LEN
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_MESSAGE_SUBID
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_MESSAGE_TS
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_OFFLINE_SET
 - BPNGDefines.h, [85](#)

BPNGDefines.h, 85
 BPNG_INVALID_RPC_COMMAND
 BPNGDefines.h, 86
 BPNG_INVALID_TSL_CASCDING
 BPNGDefines.h, 86
 BPNG_INVALID_TSL_CLUSTER
 BPNGDefines.h, 86
 BPNG_LOCAL_PATH_NOT_FOUND
 BPNGDefines.h, 85
 BPNG_LOGGER_NOT_FOUND
 BPNGDefines.h, 84
 BPNG_MIN_VERSION_ERROR
 BPNGDefines.h, 86
 BPNG_NO_TRACE_DATA_ON_DOWNLOAD_STOPPING
 BPNGDefines.h, 86
 BPNG_NOERR
 BPNGDefines.h, 84
 BPNG_NOT_CONNECTED
 BPNGDefines.h, 84
 BPNG_NOTHING_TO_CONVERT
 BPNGDefines.h, 85
 BPNG_NOTHING_TO_DOWNLOAD
 BPNGDefines.h, 85
 BPNG_NOTHING_TO_TEST_REPORT
 BPNGDefines.h, 86
 BPNG_PARAMETER_MISMATCH
 BPNGDefines.h, 85
 BPNG_RDB_OPEN_FAILED
 BPNGDefines.h, 85
 BPNG_RDB_SQLITE_QUERY_ERROR
 BPNGDefines.h, 85
 BPNG_READ_LOCAL_FILE_ERROR
 BPNGDefines.h, 85
 BPNG_TARGET_PATH_TOO_LONG
 BPNGDefines.h, 85
 BPNG_TMPBUS_COPYRDB_ERROR
 BPNGDefines.h, 84
 BPNG_TMPBUS_NOT_CONNECTED
 BPNGDefines.h, 84
 BPNG_TMPBUS_REQUEST_ERROR
 BPNGDefines.h, 85
 BPNG_TMPBUS_SEND_MSG_ERROR
 BPNGDefines.h, 84
 BPNG_TMT_FILE_ID_ERROR
 BPNGDefines.h, 85
 BPNG_TMT_FORMAT_ERROR_TS
 BPNGDefines.h, 85
 BPNG_TMT_FORMAT_ERROR_VERSION
 BPNGDefines.h, 85
 BPNG_UNSPECIFIED_ERROR
 BPNGDefines.h, 86
 BPNG_USER_CANCELLED
 BPNGDefines.h, 85
 BPNG_WARNING_CLOSE_TRACE_FILES

BPNGDefines.h, 86
 BPNG_WARNING_MESSAGES_NOT_CONVERTED
 BPNGDefines.h, 86
 BPNG_WARNING_NO_ESO_TRACE
 BPNGDefines.h, 86
 BPNG_WARNING_RECOVERING_FAILED
 BPNGDefines.h, 87
 BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONES
 BPNGDefines.h, 86
 BPNG_WRITE_LOCAL_FILE_ERROR
 BPNGDefines.h, 85
 BPNG_XML_PARSER_ERROR
 BPNGDefines.h, 85
 BPNG_ZIP_EXCEEDS_FATFS_MAX
 BPNGDefines.h, 85
 BPNGBugreportMode
 BPNGDefines.h, 83
 BPNGDefines.h, 79
 BPNG_AMBIGUOUS_IP, 84
 BPNG_CONNECT_FTP_FAILED, 84
 BPNG_CONNECT_TMPBUS_FAILED, 84
 BPNG_CONV_FORMAT_ERROR, 85
 BPNG_CONV_SET_NOT_FOUND, 85
 BPNG_CONVERSION_ERRORS, 85
 BPNG_DIR_EXISTS_ERROR, 85
 BPNG_DOWNLOAD_ERRORS, 85
 BPNG_ENGLISH, 87
 BPNG_EXCEPTION, 86
 BPNG_EXCEPTION_MINI_DUMP, 86
 BPNG_FAILED_TO_CONNECT_STREAMING,
 84
 BPNG_FAILED_TO_CREATE_LOCAL_FILE_OR_DIRECTORY,
 85
 BPNG_FILE_EXISTS_ERROR, 85
 BPNG_FTP_CHANGE_CWD_ERROR, 84
 BPNG_FTP_CREATE_REMOTE_DIR_ERROR,
 84
 BPNG_FTP_LOGIN_FAILED, 84
 BPNG_FTP_NOT_CONNECTED, 84
 BPNG_FTP_READ_REMOTE_FILE_ERROR,
 84
 BPNG_FTP_REMOTE_PATH_NOT_FOUND,
 84
 BPNG_FTP_REMOVE_REMOTE_DIR_ERROR,
 84
 BPNG_FTP_REMOVE_REMOTE_FILE_ERROR,
 84
 BPNG_FTP_SERVER_NOT_FOUND, 84
 BPNG_FTP_TRANSFER_USER_CANCELED,
 84
 BPNG_FTP_WRITE_REMOTE_FILE_ERROR,
 84
 BPNG_FW_VERSION_CHECK_ERROR, 85
 BPNG_GERMAN, 87

- BPNG_INCOMPATIBLE_RDB, 86
- BPNG_INCONSISTENT_TSL_FWVERSIONS, 86
- BPNG_INITIALISATION_ERROR, 85
- BPNG_INVALID_MESSAGE_ERROR, 85
- BPNG_INVALID_MESSAGE_ID, 85
- BPNG_INVALID_MESSAGE_LEN, 85
- BPNG_INVALID_MESSAGE_SUBID, 85
- BPNG_INVALID_MESSAGE_TS, 85
- BPNG_INVALID_OFFLINE_SET, 85
- BPNG_INVALID_RPC_COMMAND, 86
- BPNG_INVALID_TSL_CASCDING, 86
- BPNG_INVALID_TSL_CLUSTER, 86
- BPNG_LOCAL_PATH_NOT_FOUND, 85
- BPNG_LOGGER_NOT_FOUND, 84
- BPNG_MIN_VERSION_ERROR, 86
- BPNG_NO_TRACE_DATA_ON_DOWNLOAD_STATION, 86
- BPNG_NOERR, 84
- BPNG_NOT_CONNECTED, 84
- BPNG_NOTHING_TO_CONVERT, 85
- BPNG_NOTHING_TO_DOWNLOAD, 85
- BPNG_NOTHING_TO_TEST_REPORT, 86
- BPNG_PARAMETER_MISMATCH, 85
- BPNG_RDB_OPEN_FAILED, 85
- BPNG_RDB_SQLITE_QUERY_ERROR, 85
- BPNG_READ_LOCAL_FILE_ERROR, 85
- BPNG_TARGET_PATH_TOO_LONG, 85
- BPNG_TMPBUS_COPYRDB_ERROR, 84
- BPNG_TMPBUS_NOT_CONNECTED, 84
- BPNG_TMPBUS_REQUEST_ERROR, 85
- BPNG_TMPBUS_SEND_MSG_ERROR, 84
- BPNG_TMT_FILE_ID_ERROR, 85
- BPNG_TMT_FORMAT_ERROR_TS, 85
- BPNG_TMT_FORMAT_ERROR_VERSION, 85
- BPNG_UNSPECIFIED_ERROR, 86
- BPNG_USER_CANCELLED, 85
- BPNG_WARNING_CLOSE_TRACE_FILES, 86
- BPNG_WARNING_MESSAGES_NOT_CONVERTED, 86
- BPNG_WARNING_NO_ESO_TRACE, 86
- BPNG_WARNING_RECOVERING_FAILED, 87
- BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONE, 86
- BPNG_WRITE_LOCAL_FILE_ERROR, 85
- BPNG_XML_PARSER_ERROR, 85
- BPNG_ZIP_EXCEEDS_FATFS_MAX, 85
- BPNGBugreportMode, 83
- BPNGDeviceType, 83
- BPNGErrCode, 84
- BPNGLoggerStatus, 86
- BPNGWarningCode, 86
- BR_FDB_RDB, 83
- BR_FULL_ALL_TRACES, 83
- BR_FULL_TIMESPAN_TRACES, 83
- BR_FULL_WO_TRACES, 83
- BR_ONLY_CLIENT, 83
- BR_ONLY_LOGS, 83
- CH_ANALOG_IN, 87
- CH_CAMERA, 87
- CH_CAN, 87
- CH_CCPXCP, 87
- CH_DIAG, 87
- CH_DIGITAL_IN, 87
- CH_ECL, 87
- CH_ETHERNET, 87
- CH_FLEXRAY, 87
- CH_GPS, 87
- CH_LIN, 87
- CH_MII, 87
- CH_MOST150_CTRL, 87
- CH_MOST150_MDP, 87
- CH_MOST150_MEP, 87
- CH_MOST150_STREAM, 87
- CH_MOST25_CTRL, 87
- CH_MOST25_MDP, 87
- CH_MOST25_SYNC, 87
- CH_SERIAL, 87
- CH_TTY, 87
- CH_UNDEFINED, 87
- ChannelType, 87
- DEV_BP2, 83
- DEV_BP2_V1X, 83
- DEV_BP2_V2X, 83
- DEV_BP_REMOTE, 84
- DEV_BP_TOUCH, 84
- DEV_BPMINI, 83
- DEV_RAPID, 84
- DEV_RC_TOUCH, 84
- DEV_TRACE_COLL, 84
- DEV_TRACE_COLL_DS, 84
- DEV_TSL, 84
- LS_ERROR, 86
- LS_FWUPDATE, 86
- LS_MEM, 86
- LS_NOSYNC, 86
- LS_ZONE, 86
- LS_RING, 86
- LS_WARNING, 86
- LanguageID, 87
- OBSOLETE_CH_CANLS, 87
- R_BIT_MASK_OVERLAP, 88
- R_FORBIDDEN_TAB_VALUE, 88
- R_UNKNOWN, 88
- R_UNSUPPORTED_BIT_MASK, 88

- R_UNSUPPORTED_COMPU_TAB, 88
- Reason, 87
- BPNGDeviceType
 - BPNGDefines.h, 83
- BPNGErrCode
 - BPNGDefines.h, 84
- BPNGError, 25
- BPNGLoggerDetector, 25
 - BPNGLoggerDetector, 27
 - getLoggerList, 27
 - getOverwritingPermission, 27
 - getTSLs, 27
 - onBPNGDeviceDetected, 27
 - onBPNGDeviceDisappeared, 28
 - onBPNGDeviceStateChange, 28
 - onConversionStart, 28
 - onCriticalDiskSpace, 28
 - onDataRecoverProgress, 28
 - onDownloadStart, 29
 - onExtractionPasswordRequired, 29
 - onGetLogReportProgress, 29
 - onInvalidPwConfigFound, 29
 - onLogInDataRequired, 29
 - onProgressConversion, 30
 - onProgressDataDownload, 30, 31
 - onProgressDeletion, 31
 - onStatusMessage, 31
 - onTargetPathTooLong, 31
 - onWarning, 32
- BPNGLoggerDetector.hh, 88
- BPNGLoggerStatus
 - BPNGDefines.h, 86
- BPNGWarningCode
 - BPNGDefines.h, 86
- BR_FDB_RDB
 - BPNGDefines.h, 83
- BR_FULL_ALL_TRACES
 - BPNGDefines.h, 83
- BR_FULL_TIMESPAN_TRACES
 - BPNGDefines.h, 83
- BR_FULL_WO_TRACES
 - BPNGDefines.h, 83
- BR_ONLY_CLIENT
 - BPNGDefines.h, 83
- BR_ONLY_LOGS
 - BPNGDefines.h, 83
- CH_ANALOG_IN
 - BPNGDefines.h, 87
- CH_CAMERA
 - BPNGDefines.h, 87
- CH_CAN
 - BPNGDefines.h, 87
- CH_CCPXCP
 - BPNGDefines.h, 87
- CH_DIAG
 - BPNGDefines.h, 87
- CH_DIGITAL_IN
 - BPNGDefines.h, 87
- CH_ECL
 - BPNGDefines.h, 87
- CH_ETHERNET
 - BPNGDefines.h, 87
- CH_FLEXRAY
 - BPNGDefines.h, 87
- CH_GPS
 - BPNGDefines.h, 87
- CH_LIN
 - BPNGDefines.h, 87
- CH_MII
 - BPNGDefines.h, 87
- CH_MOST150_CTRL
 - BPNGDefines.h, 87
- CH_MOST150_MDP
 - BPNGDefines.h, 87
- CH_MOST150_MEP
 - BPNGDefines.h, 87
- CH_MOST150_STREAM
 - BPNGDefines.h, 87
- CH_MOST25_CTRL
 - BPNGDefines.h, 87
- CH_MOST25_MDP
 - BPNGDefines.h, 87
- CH_MOST25_SYNC
 - BPNGDefines.h, 87
- CH_SERIAL
 - BPNGDefines.h, 87
- CH_TTY
 - BPNGDefines.h, 87
- CH_UNDEFINED
 - BPNGDefines.h, 87
- CONFIG
 - RdbDefines.h, 93
- ChannelType
 - BPNGDefines.h, 87
- clearDBCFileAssignments
 - IBPNGClient, 37
- connectLogger
 - IBPNGClient, 37
- convertData
 - IBPNGClient, 37
- createEmptyOnlineLoggerInfo
 - IBPNGClient.h, 89
- createNewClientProperties
 - IBPNGClient.h, 90
- createNewConversionSet
 - IBPNGClient, 38
- createTestReport

- IBPNGClient, 38
- DATA_DELETED
 - RdbDefines.h, 92
- DEV_BP2
 - BPNGDefines.h, 83
- DEV_BP2_V1X
 - BPNGDefines.h, 83
- DEV_BP2_V2X
 - BPNGDefines.h, 83
- DEV_BP_REMOTE
 - BPNGDefines.h, 84
- DEV_BP_TOUCH
 - BPNGDefines.h, 84
- DEV_BPMINI
 - BPNGDefines.h, 83
- DEV_RAPID
 - BPNGDefines.h, 84
- DEV_RC_TOUCH
 - BPNGDefines.h, 84
- DEV_TRACE_COLL
 - BPNGDefines.h, 84
- DEV_TRACE_COLL_DS
 - BPNGDefines.h, 84
- DEV_TSL
 - BPNGDefines.h, 84
- DataSpan, 32
- deleteAllData
 - IBPNGClient, 38
- deleteSectionsByStartUplds
 - IBPNGClient, 38
- deviceType
 - OnlineLoggerInfo, 76
- downloadBugReport
 - IBPNGClient, 39
- downloadDataSpans
 - IBPNGClient, 39
- enableClientLogOutput
 - IBPNGClient, 40
- filterSignals
 - IBPNGClient, 40
- filterSignalsFromOfflineData
 - IBPNGClient, 40
- flashDeviceLED
 - IBPNGClient, 41
- freeClientPropertiesMemory
 - IBPNGClient.h, 90
- freeConversionSetMemory
 - IBPNGClient.h, 90
- getAvailableFormats
 - IBPNGClient, 41
- getBPNGClient
 - IBPNGClient.h, 90
- getClientProperties
 - IBPNGClient, 41
- getComment
 - IRdbEvent, 71
- getConfig
 - IBPNGClient, 41
- getConfigPath
 - IBPNGClient, 41
- getConversionError
 - IBPNGClient, 42
- getDeviceName
 - IBPNGClient, 42
- getDownloadError
 - IBPNGClient, 42
- getEventList
 - IBPNGClient, 42
- getFalseMeasureSignals
 - IBPNGClient, 42
- getLastError
 - IBPNGClient, 42
- getLicenses
 - IBPNGClient, 43
- getLoggerChannels
 - IBPNGClient, 43
- getLoggerList
 - BPNGLoggerDetector, 27
- getMemoryFillLevel
 - IBPNGClient, 43
- getNumConversionErrors
 - IBPNGClient, 43
- getNumDownloadErrors
 - IBPNGClient, 44
- getNumTSLMemberFromOfflineDataSet
 - IBPNGClient.h, 90
- getOverwritingPermission
 - BPNGLoggerDetector, 27
 - IBPNGClientListener, 54
- getPwdFile
 - IBPNGClient, 44
- getReferenceDataBasePath
 - IBPNGClient, 44
- getTSLs
 - BPNGLoggerDetector, 27
- getTimeZone
 - IRdbEvent, 71
- getTraceBlockList
 - IBPNGClient, 44
- getUniqueld
 - IRdbEvent, 71
- getVersions
 - IBPNGClient, 45

- IBPNGClient, 32
 - activateGatewayLoggerDetection, 36
 - assignDBCFile, 37
 - assignDatabaseFile, 36
 - clearDBCFileAssignments, 37
 - connectLogger, 37
 - convertData, 37
 - createNewConversionSet, 38
 - createTestReport, 38
 - deleteAllData, 38
 - deleteSectionsByStartUpIds, 38
 - downloadBugReport, 39
 - downloadDataSpans, 39
 - enableClientLogOutput, 40
 - filterSignals, 40
 - filterSignalsFromOfflineData, 40
 - flashDeviceLED, 41
 - getAvailableFormats, 41
 - getClientProperties, 41
 - getConfig, 41
 - getConfigPath, 41
 - getConversionError, 42
 - getDeviceName, 42
 - getDownloadError, 42
 - getEventList, 42
 - getFalseMeasureSignals, 42
 - getLastError, 42
 - getLicenses, 43
 - getLoggerChannels, 43
 - getMemoryFillLevel, 43
 - getNumConversionErrors, 43
 - getNumDownloadErrors, 44
 - getPwdFile, 44
 - getReferenceDataBasePath, 44
 - getTraceBlockList, 44
 - getVersions, 45
 - initialize, 45
 - isPasswordProtectionSupported, 45
 - keepLoggerAlive, 45
 - reconfigLogger, 47
 - release, 47
 - removeAllLicenses, 48
 - restartDevice, 48
 - scanNetworkForLogger, 48
 - setClientProperties, 48
 - setDefaultConfig, 48
 - setDevice, 49
 - setInfoEvent, 49
 - setMarker, 49
 - setOfflineData, 49
 - setPwdFile, 49
 - setTSLCluster, 51
 - setTime, 51
 - shutdownDevice, 51
 - startLiveDownload, 51
 - synchronizeRdb, 52
 - updateFirmware, 52
 - updateLicenses, 52
- IBPNGClient.h, 88
 - addLogListener, 89
 - createEmptyOnlineLoggerInfo, 89
 - createNewClientProperties, 90
 - freeClientPropertiesMemory, 90
 - freeConversionSetMemory, 90
 - getBPNGClient, 90
 - getNumTSLMemberFromOfflineDataSet, 90
 - writeLogFile, 90
 - writeLogToCout, 90
- IBPNGClientListener, 53
 - getOverwritingPermission, 54
 - onBPNGDeviceDetected, 54
 - onBPNGDeviceDisappeared, 54
 - onBPNGDeviceStateChange, 54
 - onConversionStart, 54
 - onCriticalDiskSpace, 55
 - onDataRecoverProgress, 55
 - onDownloadStart, 55
 - onExtractionPasswordRequired, 55
 - onGetLogReportProgress, 56
 - onInvalidPwConfigFound, 56
 - onLogInDataRequired, 56
 - onProgressConversion, 56
 - onProgressDataDownload, 57
 - onProgressDeletion, 57
 - onStatusMessage, 58
 - onTargetPathTooLong, 58
 - onWarning, 58
- IBPNGClientListener.h, 91
- IChannel, 58
- IChannelList, 59
- IClientProperties, 59
 - addAnalogPortSettings, 63
 - addDigitalPortSettings, 64
 - setCANPseudoMsgTimeStampProperties, 64
 - setCANPseudoMsgTriggerProperties, 64
 - setCommonProperties, 65
 - setFlexRayPseudoMsgProperties, 66
 - setMOSTPseudoMsgProperties, 66
- IClientProperties.h, 91
- IConversionSet, 66
 - addChannel, 67
 - addRdbldRange, 68
 - addTimeSpan, 68
 - loadConversionFilters, 68
- IFalseMeasureSignal, 68
- IFalseMeasureSignalList, 69
- IFormatInfo, 69
- IFormatList, 70

- INFO
 - RdbDefines.h, [92](#)
- IRdbEvent, [70](#)
 - getComment, [71](#)
 - getTimeZone, [71](#)
 - getUniqueId, [71](#)
- IRdbEventList, [71](#)
- IRdbTraceBlock, [72](#)
- IRdbTraceBlockList, [72](#)
- ITesttoolsChannel, [72](#)
- ITesttoolsChannelList, [73](#)
- initialize
 - IBPNGClient, [45](#)
- isPasswordProtectionSupported
 - IBPNGClient, [45](#)
- keepLoggerAlive
 - IBPNGClient, [45](#)
- LS_ERROR
 - BPNGDefines.h, [86](#)
- LS_FWUPDATE
 - BPNGDefines.h, [86](#)
- LS_MEM
 - BPNGDefines.h, [86](#)
- LS_NOSYNC
 - BPNGDefines.h, [86](#)
- LS_OK
 - BPNGDefines.h, [86](#)
- LS_RING
 - BPNGDefines.h, [86](#)
- LS_WARNING
 - BPNGDefines.h, [86](#)
- LanguageID
 - BPNGDefines.h, [87](#)
- loadConversionFilters
 - IConversionSet, [68](#)
- LogInData, [74](#)
- loggerStatus
 - OnlineLoggerInfo, [76](#)
- MARKER
 - RdbDefines.h, [92](#)
- MemoryFillLevel, [74](#)
- NEW_TIME
 - RdbDefines.h, [92](#)
- OBSOLETE_CH_CANLS
 - BPNGDefines.h, [87](#)
- onBPNGDeviceDetected
 - BPNGLoggerDetector, [27](#)
 - IBPNGClientListener, [54](#)
- onBPNGDeviceDisappeared
 - BPNGLoggerDetector, [28](#)
- IBPNGClientListener, [54](#)
- onBPNGDeviceStateChange
 - BPNGLoggerDetector, [28](#)
 - IBPNGClientListener, [54](#)
- onConversionStart
 - BPNGLoggerDetector, [28](#)
 - IBPNGClientListener, [54](#)
- onCriticalDiskSpace
 - BPNGLoggerDetector, [28](#)
 - IBPNGClientListener, [55](#)
- onDataRecoverProgress
 - BPNGLoggerDetector, [28](#)
 - IBPNGClientListener, [55](#)
- onDownloadStart
 - BPNGLoggerDetector, [29](#)
 - IBPNGClientListener, [55](#)
- onExtractionPasswordRequired
 - BPNGLoggerDetector, [29](#)
 - IBPNGClientListener, [55](#)
- onGetLogReportProgress
 - BPNGLoggerDetector, [29](#)
 - IBPNGClientListener, [56](#)
- onInvalidPwConfigFound
 - BPNGLoggerDetector, [29](#)
 - IBPNGClientListener, [56](#)
- onLogInDataRequired
 - BPNGLoggerDetector, [29](#)
 - IBPNGClientListener, [56](#)
- onProgressConversion
 - BPNGLoggerDetector, [30](#)
 - IBPNGClientListener, [56](#)
- onProgressDataDownload
 - BPNGLoggerDetector, [30, 31](#)
 - IBPNGClientListener, [57](#)
- onProgressDeletion
 - BPNGLoggerDetector, [31](#)
 - IBPNGClientListener, [57](#)
- onStatusMessage
 - BPNGLoggerDetector, [31](#)
 - IBPNGClientListener, [58](#)
- onTargetPathTooLong
 - BPNGLoggerDetector, [31](#)
 - IBPNGClientListener, [58](#)
- onWarning
 - BPNGLoggerDetector, [32](#)
 - IBPNGClientListener, [58](#)
- OnlineLoggerInfo, [75](#)
 - deviceType, [76](#)
 - loggerStatus, [76](#)
- OnlineLoggerInfoStringPair, [76](#)
- R_BIT_MASK_OVERLAP
 - BPNGDefines.h, [88](#)
- R_FORBIDDEN_TAB_VALUE

- BPNGDefines.h, 88
- R_UNKNOWN
 - BPNGDefines.h, 88
- R_UNSUPPORTED_BIT_MASK
 - BPNGDefines.h, 88
- R_UNSUPPORTED_COMPU_TAB
 - BPNGDefines.h, 88
- RdbDefines.h, 92
 - CONFIG, 93
 - DATA_DELETED, 92
 - INFO, 92
 - MARKER, 92
 - NEW_TIME, 92
 - RdbEventType, 92
 - SHUTDOWN, 92
 - SLAVE_OFFSET, 92
 - SLAVE_TO_MASTER, 92
 - START_TESTDRIVE, 93
 - STARTUP, 92
 - STOP_TESTDRIVE, 93
 - SUDDEN_DEATH, 93
 - TESTDRIVE_INFO, 93
 - TIME_SET, 92
 - TSL_SESSION_END, 93
 - TSL_SESSION_START, 93
 - TSL_SLAVE_OFFSET, 93
 - TSL_SLAVE_TO_MASTER, 93
 - WAKEUP, 93
- RdbEvent2, 77
- RdbEventList, 78
- RdbEventList.hh, 93
- RdbEventType
 - RdbDefines.h, 92
- Reason
 - BPNGDefines.h, 87
- reconfigLogger
 - IBPNGClient, 47
- release
 - IBPNGClient, 47
- removeAllLicenses
 - IBPNGClient, 48
- restartDevice
 - IBPNGClient, 48
- SHUTDOWN
 - RdbDefines.h, 92
- SLAVE_OFFSET
 - RdbDefines.h, 92
- SLAVE_TO_MASTER
 - RdbDefines.h, 92
- START_TESTDRIVE
 - RdbDefines.h, 93
- STARTUP
 - RdbDefines.h, 92
- STOP_TESTDRIVE
 - RdbDefines.h, 93
- SUDDEN_DEATH
 - RdbDefines.h, 93
- scanNetworkForLogger
 - IBPNGClient, 48
- setCANPseudoMsgTimeStampProperties
 - IClientProperties, 64
- setCANPseudoMsgTriggerProperties
 - IClientProperties, 64
- setClientProperties
 - IBPNGClient, 48
- setCommonProperties
 - IClientProperties, 65
- setDefaultConfig
 - IBPNGClient, 48
- setDevice
 - IBPNGClient, 49
- setFlexRayPseudoMsgProperties
 - IClientProperties, 66
- setInfoEvent
 - IBPNGClient, 49
- setMOSTPseudoMsgProperties
 - IClientProperties, 66
- setMarker
 - IBPNGClient, 49
- setOfflineData
 - IBPNGClient, 49
- setPwdFile
 - IBPNGClient, 49
- setTSLCluster
 - IBPNGClient, 51
- setTime
 - IBPNGClient, 51
- shutdownDevice
 - IBPNGClient, 51
- startLiveDownload
 - IBPNGClient, 51
- synchronizeRdb
 - IBPNGClient, 52
- TESTDRIVE_INFO
 - RdbDefines.h, 93
- TIME_SET
 - RdbDefines.h, 92
- TSL_SESSION_END
 - RdbDefines.h, 93
- TSL_SESSION_START
 - RdbDefines.h, 93
- TSL_SLAVE_OFFSET
 - RdbDefines.h, 93
- TSL_SLAVE_TO_MASTER
 - RdbDefines.h, 93
- TSLCluster, 78

- updateFirmware
 - IBPNGClient, [52](#)
- updateLicenses
 - IBPNGClient, [52](#)
- WAKEUP
 - RdbDefines.h, [93](#)
- writeLogFile
 - IBPNGClient.h, [90](#)
- writeLogToCout
 - IBPNGClient.h, [90](#)