



BLUEPIRAT Client Library 5.1.0.8

User's manual

Generated by Doxygen 1.8.0

Tue Nov 2 2021 11:47:10



# Contents

<b>1</b>	<b>User's manual - BLUEPIRAT Client Library 5.1.0.8</b>	<b>1</b>
1.1	General . . . . .	1
1.2	Functionality . . . . .	1
1.3	Thread safety . . . . .	2
1.4	Demo project . . . . .	2
<b>2</b>	<b>Deprecated List</b>	<b>21</b>
<b>3</b>	<b>Hierarchical Index</b>	<b>23</b>
3.1	Class Hierarchy . . . . .	23
<b>4</b>	<b>Class Index</b>	<b>25</b>
4.1	Class List . . . . .	25
<b>5</b>	<b>File Index</b>	<b>27</b>
5.1	File List . . . . .	27
<b>6</b>	<b>Class Documentation</b>	<b>29</b>
6.1	BPNGError Struct Reference . . . . .	29
6.2	BPNGLoggerDetector Class Reference . . . . .	29
6.3	DataSpan Struct Reference . . . . .	37
6.4	IBPNGClient Struct Reference . . . . .	37
6.5	IBPNGClientListener Struct Reference . . . . .	59
6.6	IChannel Struct Reference . . . . .	66
6.7	IChannelList Struct Reference . . . . .	66
6.8	IClientProperties Struct Reference . . . . .	67
6.9	IConversionSet Struct Reference . . . . .	75
6.10	IFalseMeasureSignal Struct Reference . . . . .	77
6.11	IFalseMeasureSignalList Struct Reference . . . . .	78
6.12	IFormatInfo Struct Reference . . . . .	78
6.13	IFormatList Struct Reference . . . . .	79
6.14	IRdbEvent Struct Reference . . . . .	79
6.15	IRdbEventList Struct Reference . . . . .	80
6.16	IRdbTraceBlock Struct Reference . . . . .	81
6.17	IRdbTraceBlockList Struct Reference . . . . .	81
6.18	ITesttoolsChannel Struct Reference . . . . .	81
6.19	ITesttoolsChannelList Struct Reference . . . . .	82
6.20	LogInData Struct Reference . . . . .	83
6.21	MemoryFillLevel Struct Reference . . . . .	83
6.22	OnlineLoggerInfo Struct Reference . . . . .	84
6.23	OnlineLoggerInfoStringPair Struct Reference . . . . .	86
6.24	OnlineLoggerInfoWrapper Class Reference . . . . .	86
6.25	RdbEvent2 Struct Reference . . . . .	88

6.26 RdbEventList Class Reference . . . . .	88
6.27 RdbTraceBlock2 Struct Reference . . . . .	89
6.28 RdbTraceBlockList Class Reference . . . . .	90
6.29 TSLCluster Struct Reference . . . . .	91
6.30 TSLClusterImpl Class Reference . . . . .	91
<b>7 File Documentation</b>	<b>95</b>
7.1 BPNGDefines.h File Reference . . . . .	95
7.2 BPNGLoggerDetector.hh File Reference . . . . .	106
7.3 clientUtil.hh File Reference . . . . .	107
7.4 IBPNGClient.h File Reference . . . . .	110
7.5 IBPNGClientListener.h File Reference . . . . .	113
7.6 IClientProperties.h File Reference . . . . .	113
7.7 OnlineLoggerInfoWrapper.hh File Reference . . . . .	114
7.8 RdbDefines.h File Reference . . . . .	114
7.9 RdbEventList.hh File Reference . . . . .	116
7.10 RdbTraceBlockList.hh File Reference . . . . .	116
7.11 TSLClusterImpl.hh File Reference . . . . .	117
<b>Index</b>	<b>119</b>

## Chapter 1

# User's manual - BLUEPIRAT Client Library 5.1.0.8

### 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 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!

## 1.4 Demo project

The "sample" directory contains a demo project for the BLUEPIRAT Client Library.

Exampe for lib unsage:

```
//*****
//
// BPNGClientLibTest.cc
//
//*****

// sys
#include <algorithm>
#include <chrono>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <dirent.h>
#include <getopt.h>
#include <iostream>
#include <iomanip>
#include <set>
#include <sstream>
#include <string>
#include <unistd.h>
#include <utility>
#include <vector>

// tmlib
// following two includes will be removed for clientlib release
#include <utility.hh>
#include <fileutils.hh>

// atom
#include <RdbDefines.h>

// client
#include <BPNGDefines.h>
#include <IBPNGClient.h>

// test
#include "BPNGLoggerDetector.hh"
#include "RdbEventList.hh"
#include "RdbTraceBlockList.hh"
#include "clientUtil.hh"

using namespace std;
using namespace std::chrono;

const char DEFAULT_TARGET_IP[] = "192.168.0.233";
const char DEFAULT_IP[] = "0.0.0.0";

enum ConfigMode
{
    CONFIG_NONE = 0,

    CONFIG_GET = 0x01,
    CONFIG_SET = 0x02,
    CONFIG_DEFAULT = 0x04,

    CONFIG_ALL = CONFIG_GET | CONFIG_SET | CONFIG_DEFAULT
};

static ConfigMode toConfig(const char *arg)
{

```

```

    if (!strcmp(arg, "get"))
        return CONFIG_GET;
    else if (!strcmp(arg, "set"))
        return CONFIG_SET;
    else if (!strcmp(arg, "default"))
        return CONFIG_DEFAULT;
    else if (!strcmp(arg, "all"))
        return CONFIG_ALL;
    else
        return CONFIG_NONE;
}

static BPNGBugreportMode toBugreport(const char *arg)
{
    if (!strcmp(arg, "full"))
        return BR_FULL_WO_TRACES;
    else if (!strcmp(arg, "logs"))
        return BR_ONLY_LOGS;
    else if (!strcmp(arg, "client"))
        return BR_ONLY_CLIENT;
    else if (!strcmp(arg, "db"))
        return BR_FDB_RDB;
    else if (!strcmp(arg, "all"))
        return BR_FULL_ALL_TRACES;
    else if (!strcmp(arg, "traces"))
        return BR_FULL_TIMESPAN_TRACES;
    else
        return BR_FULL_WO_TRACES;
}

static vector<string> readZipsFromDirectory(string dir)
{
    vector<string> output;
    DIR* 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 string getLocalDateString()
{
    time_t t;
    tm tt;
    ostringstream buffer;

    time(&t);
    localtime_r(&t, &tt);
    buffer << std::put_time(&tt, "%F_%H-%M-%S");

    return buffer.str();
}

static string unixtimeToLocaltime(uint64_t unixtime)
{
    time_t t;
    tm tt;

    ostringstream buffer;

    t = time_t(unixtime / 1000000UL);
    localtime_r(&t, &tt);
    buffer << std::put_time(&tt, "%F_%H-%M-%S");

    return buffer.str();
}

```

```

/*****
 *      Client function
 *****/
static void checkForTslNeighbor(const string &ip, const string &ipNeighbor,
    const BPNGLoggerDetector &listener,
    TSLClusterImpl &tsl)
{
    OnlineLoggerInfo oli = listener.getLoggerInfoForIP(ipNeighbor.c_str());
    //    cout << "OnlineLoggerInfo: " << oli << endl;

    string eth0 = oli.tslEth0IP;
    string eth1 = oli.tslEth1IP;

    if (ip != DEFAULT_IP && (eth0 == ip || eth1 == ip))
        tsl.addDevice(oli);

    if (eth0 != ip && eth0 != DEFAULT_IP && !eth0.empty())
        checkForTslNeighbor(ipNeighbor, oli.tslEth0IP, listener, tsl);

    if (eth1 != ip && eth1 != DEFAULT_IP && !eth1.empty())
        checkForTslNeighbor(ipNeighbor, oli.tslEth1IP, listener, tsl);
}

static TSLClusterImpl getTslNetwork(const char *ip, const
    BPNGLoggerDetector &listener)
{
    cout << __func__ << ": ip=" << ip << endl;

    OnlineLoggerInfo oli = listener.getLoggerInfoForIP(ip);
    TSLClusterImpl cluster(oli);

    cout << "OnlineLoggerInfo: " << oli << endl;

    string eth0 = oli.tslEth0IP;
    string eth1 = oli.tslEth1IP;
    if (eth0 != DEFAULT_IP && !eth0.empty())
        checkForTslNeighbor(ip, eth0, listener, cluster);

    if (eth1 != DEFAULT_IP && !eth1.empty())
        checkForTslNeighbor(ip, eth1, listener, cluster);

    return cluster;
}

static bool connectInitLogger(const char *module, IBPNGClient *bpngClient,
    TSLClusterImpl tsl, bool initialize = true)
{
    cout << module << ": connect Logger... ";
    bool ret = false;
    if (tsl.getTSLSize() > 1)
    {
        bpngClient->setTSLCluster(tsl.getTSLCluster());
        ret = bpngClient->connect();
    }
    else
    {
        ret = bpngClient->connectLogger(1, &tsl.getTSLCluster().loggerArray[0]);
    }

    if (ret)
    {
        cout << "OK" << endl;
    }
    else
    {
        BPNGError err = bpngClient->getLastError();
        cout << "Failed to connect tsl. " << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        bpngClient->release();
        return false;
    }

    if (!initialize)
        return true;

    cout << "init online... ";
    if (bpngClient->initialize())
    {

```



```

        cout << "OK" << endl;
    }
    else
    {
        BPNGError err = bpngClient->getLastError();
        cerr << "init online failed: " << err.msg << endl;
        bpngClient->disconnectLogger();
        return false;
    }

    return true;
}

static void setTimeTest(TSLClusterImpl tsl)
{
    IBPNGClient *bpng = getBPNGClient();

    if (!connectInitLogger(__func__, bpng, tsl))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." << endl;
        return;
    }

    for (int i = 0; i < 5; i++)
    {
        auto tnow = system_clock::now() + system_clock::duration(seconds(i * 100));
        if (bpng->setTime(system_clock::to_time_t(tnow)))
            cout << "loop " << i << ": settime ok test time " << tnow << " ok" << endl;
        else
            cerr << "loop " << i << ": settime to test time " << tnow << " failed" << endl;

        auto tget = system_clock::from_time_t(bpng->getCurrentLoggerTime());
        cout << "loop " << i << ": returned " << tget << endl;

        sleep(5);
    }

    auto tnow = system_clock::now();
    if (bpng->setTime(system_clock::to_time_t(tnow)))
        cout << "settime to correct time ok" << endl;
    else
        cerr << "settime to correct time failed" << endl;

    cout << "Test finished" << endl;
    bpng->disconnectLogger();
}

static void getTimeTest(TSLClusterImpl tsl)
{
    IBPNGClient *bpng = getBPNGClient();

    cout << "bpng=" << bpng << endl;

    if (!connectInitLogger(__func__, bpng, tsl))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." << endl;
        return;
    }

    for (int i = 0; i < 4; i++)
    {
        auto tget = system_clock::from_time_t(bpng->getCurrentLoggerTime());
        cout << "loop " << i << ": get " << tget << endl;
        sleep(5);
    }

    bpng->disconnectLogger();
}

static void configTest(TSLClusterImpl tsl, const char *path, const vector<ConfigMode> &config)
{
    IBPNGClient *bpng = getBPNGClient();

    if (!connectInitLogger(__func__, bpng, tsl))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." << endl;
    }
}

```

```

    return;
}

for (auto c : config)
{
    if (c & CONFIG_GET)
    {
        ostringstream targetPath;
        targetPath << path << "/Config_";
        if (tsl.getTSLSize() == 1)
        {
            targetPath << tsl.getTSLCluster().loggerArray[0].name << '_';
            targetPath << tsl.getTSLCluster().loggerArray[0].mbnr;
        }
        else
        {
            targetPath << "bPTSL_" << tsl.getTSLName();
        }
        targetPath << '_' << getLocalDateString();

        // get and save current config
        cout << "configTest: get and save config...";
        if (!bpng->getConfig(targetPath.str().c_str()))
        {
            BPNGError err = bpng->getLastError();
            cerr << "download configuration failed: " << err.msg << endl;
            bpng->disconnectLogger();
            return;
        }
    }

    if (c & CONFIG_DEFAULT)
    {
        // setting the default config
        cout << "configTest: set default config...";
        if (!bpng->setDefaultConfig())
        {
            BPNGError err = bpng->getLastError();
            cerr << "default config failed: " << err.msg << endl;
            bpng->disconnectLogger();
            return;
        }
    }

    if (c & CONFIG_SET)
    {
        // FIXME:
        // Logger does not load the new config.

        // Logger will only accept config zip files with format:
        // bpng-*_date_time.zip

        // 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
        vector<string> configZips = readZipsFromDirectory(path);

        vector<OnlineLoggerInfoStringPair> pairs;
        vector<string> paths;

        for (const auto &iter : tsl)
        {
            for (size_t i = 0; i != configZips.size(); ++i)
            {
                string configZip = configZips[i];
                if (configZip.find(iter.mbnr) != string::npos)
                {
                    cout << configZips[i] << endl;
                    OnlineLoggerInfoStringPair pair;
                    pair.key = iter;
                    ostringstream zipPath;
                    zipPath << path << '/' << configZip;
                    paths.push_back(zipPath.str());
                    pair.value = paths.back().c_str();
                    cout << "pair.value: " << pair.value << endl;
                    pairs.push_back(pair);
                    break;
                }
            }
        }
    }
}

```

```

    }
}

// We use the same config that we downloaded
cout << "configTest: restore original config...";
if (!bpng->reconfigLogger(pairs.size(), &pairs[0]))
{
    BPNGError err = bpng->getLastError();
    cerr << "reconfiguring logger failed: " << err.msg << endl;
    cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
    bpng->disconnectLogger();
    return;
}
}

// disconnect
bpng->disconnectLogger();
}

static void bugreportTest(TSLClusterImpl tsl, const char *path,
    BPNGBugreportMode mode)
{
    IBPNGClient *bpng = getBPNGClient();

    if (!connectInitLogger(__func__, bpng, tsl))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." <<
        endl;
        return;
    }

    uint64_t startTime = 0;
    uint64_t endTime = 0;

    if (mode == BR_FULL_ALL_TRACES)
    {
        // get start and end time of all traces
        cout << "get trace block table..." << endl;
        IRdbTraceBlockList *traceBlocklist = bpng->
        getTraceBlockList();
        RdbTraceBlockList blocks(traceBlocklist);
        blocks.sortTraceBlocksById();

        startTime = blocks.begin()->m_DataStartTimeUTC;
        endTime = blocks.end()->m_DataEndTimeUTC;
    }

    string destination = "/tmp";
    char fullPath[PATH_MAX];
    char *resultPath = realpath(path, fullPath);
    if (resultPath)
        destination = fullPath;

    // name of Bugreport file
    string zipName = "Bugreport_";
    if (tsl.getTSLSize() > 1)
        zipName += tsl.getTSLName();
    else
        zipName += tsl.begin()->name;
    destination += "/";
    destination += zipName;

    int result = bpng->downloadBugReport(destination.c_str(), mode, startTime, endTime);
    if (result == 0)
    {
        BPNGError err = bpng->getLastError();
        cerr << "download Bugreport failed: " << err.msg << endl;
    }
    else if (result < 0)
    {
        cerr << "download Bugreport failed: user aborted erase!" << endl;
    }
    else
    {
        cout << "download Bugreport successfull finished. Location: " << destination << endl;
    }

    // disconnect

```

```

    bpng->disconnectLogger();
}

static void downloadTest(TSLClusterImpl tsl, const char *path, bool all = false)
{
    IBPNGClient *bpngClient = getBPNGClient();

    string loggername = (tsl.getTSLSize() == 1) ? tsl.begin()->name : tsl.
        getTSLName();

    if (!connectInitLogger(__func__, bpngClient, tsl))
    {
        cerr << "Failed to initialize logger connection to " << loggername << ", abort." << endl;
        return;
    }

    uint64_t startId = 0;
    cout << "get events..." << endl;
    IRdbEventList *list = bpngClient->getEventList();
    RdbEventList eventList(list);

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

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

    cout << "get trace block table..." << endl;
    IRdbTraceBlockList *traceBlocklist = bpngClient->
        getTraceBlockList();
    RdbTraceBlockList blocks(traceBlocklist);
    blocks.sortTraceBlocksById();

    // if you want to download several spans, put them in a vector
    vector<DataSpan> spanVec;

    // target path as output string
    ostringstream starget;
    if (all)
    {
        uint64_t max = 0;
        if (!blocks.empty())
            max = blocks.back().m_DataBaseEntryId;

        DataSpan span;
        cout << "create data span, startID = " << startId << ", endID = " << max << endl;
        span.type = 0;
        span.start = startId;
        span.end = max;

        spanVec.push_back(span);

        starget << path << '/' << loggername << '_'
            << unixtimeToLocaltime(blocks.front().m_DataStartTimeUTC) << '_'
            << unixtimeToLocaltime(blocks.back().m_DataEndTimeUTC) << ".zip";
    }
    else
    {
        // download selected time span
        // always from startup to shutdown event
        const uint64_t us = 1000000U;
        time_t startTime, endTime;
        tm starttm;
        tm endtm;

        // start time
        starttm.tm_sec = 0;
        starttm.tm_min = 0;
        starttm.tm_hour = 0;
    }
}

```

```

        starttm.tm_mday = 6;
        starttm.tm_mon = 4 - 1;
        starttm.tm_year = 2020 - 1900;
        starttm.tm_wday = 0;
        starttm.tm_yday = 0;
        starttm.tm_isdst = 0;

        // end time
        endtm.tm_sec = 59;
        endtm.tm_min = 59;
        endtm.tm_hour = 23;
        endtm.tm_mday = 6;
        endtm.tm_mon = 4 - 1;
        endtm.tm_year = 2020 - 1900;
        endtm.tm_wday = 0;
        endtm.tm_yday = 0;
        endtm.tm_isdst = 0;

        startTime = timegm(&starttm) * us;
        endTime = timegm(&endtm) * us;

        cout << "select time span: " << std::put_time(&starttm, "%F %T") << " " << std::put_time(&endtm, "%F %T %Z") << endl;

        DataSpan span;
        span.type = 0;
        span.start = 0;
        span.end = 0;

        size_t startblock = 0, endblock = 0;
        cout << "block size: " << blocks.size() << endl;
        for (size_t i = 0; i < blocks.size(); i++)
        {
            //cout << "entry id: " << blocks[i].m_DataBaseEntryId << " | timespan: " <<
            unixtimeToLocaltime(blocks[i].m_DataStartTimeUTC) << " " << unixtimeToLocaltime(blocks[i].m_DataEndTimeUTC) << endl;
            if ((blocks[i].m_DataStartTimeUTC <= startTime && blocks[i].m_DataEndTimeUTC >= startTime)
                || (i > 0 && blocks[i].m_DataStartTimeUTC >= startTime && blocks[i-1].m_DataEndTimeUTC <=
startTime)
                || (i == 0 && blocks[i].m_DataStartTimeUTC >= startTime))
            {
                span.start = blocks[i].m_DataBaseEntryId;
                startblock = i;
            }
            if ((blocks[i].m_DataStartTimeUTC <= endTime && blocks[i].m_DataEndTimeUTC >= endTime)
                || (i+1 < blocks.size() && blocks[i+1].m_DataStartTimeUTC >= endTime && blocks[i].
m_DataEndTimeUTC <= endTime)
                || (i == blocks.size() - 1 && blocks[i].m_DataEndTimeUTC <= endTime))
            {
                span.end = blocks[i].m_DataBaseEntryId;
                endblock = i;
            }
        }

        spanVec.push_back(span);
        cout << "selected id span: " << span.start << " " << span.end << endl;

        target << path << '/' << loggername << '_'
                << unixtimeToLocaltime(blocks[startblock].m_DataStartTimeUTC) << '_'
                << unixtimeToLocaltime(blocks[endblock].m_DataEndTimeUTC) << ".zip";
    }

    const string &target = target.str();

    cout << "download data to " << target << endl;
    if (!bpngClient->downloadDataSpans(spanVec.size(), &spanVec[0], target.c_str(), 0))
    {
        BPNGError err = bpngClient->getLastError();
        cerr << "download failed: " << err.msg << endl;
    }

    bpngClient->disconnectLogger();
}

static void onlineConversionTest(TSLClusterImpl tsl, const char *path, bool all)
{
    IBPNGClient *bpng = getBPNGClient();

    if (!connectInitLogger(__func__, bpng, tsl, true))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." <<

```

```

    endl;
    return;
}

IConversionSet *cs = bpng->createNewConversionSet();

const uint64_t us = 1000000U;
time_t startTime, endTime;
tm starttm;
tm endtm;

if (all)
{
    // convert all
    startTime = 0;
    endTime = time(nullptr);

    gmtime_r(&startTime, &starttm);
    gmtime_r(&endTime, &endtm);
}
else
{
    // convert specific time span (adapt this to your needs)
    starttm.tm_sec = 0;
    starttm.tm_min = 0;
    starttm.tm_hour = 0;
    starttm.tm_mday = 8;
    starttm.tm_mon = 1 - 1;
    starttm.tm_year = 2020 - 1900;
    starttm.tm_wday = 0;
    starttm.tm_yday = 0;
    starttm.tm_isdst = 0;

    endtm.tm_sec = 59;
    endtm.tm_min = 59;
    endtm.tm_hour = 23;
    endtm.tm_mday = 9;
    endtm.tm_mon = 12 - 1;
    endtm.tm_year = 2020 - 1900;
    endtm.tm_wday = 0;
    endtm.tm_yday = 0;
    endtm.tm_isdst = 1;

    startTime = mktime(&starttm);
    endTime = mktime(&endtm);
}

cout << "select time span: " << std::put_time(&starttm, "%F %T") << " ." << std::put_time(&endtm, "%F
    %T %Z") << endl;
cs->addTimeSpan(startTime * us, endTime * us);

const IChannelList *channels = bpng->getLoggerChannels();

cout << "channels = " << channels << endl;
if (!channels)
{
    cerr << "no logger channels found" << endl;
    return;
}

const auto nchannels = channels->getSize();

cout << "#channels = " << nchannels << endl;

set<ChannelType> ignoredChannels;

// With IConversionSet::addChannel() we have to add those channels to the conversion set that we want
// to convert.
// The required format is passed as third argument. Channels that should be written to the same output
// file
// must be added to IConversionSet with the same 'fileId' argument (and of course same formatId).

// In this sample we only want to convert CAN channels according to this:
// CAN #1 and CAN #2 are supposed to be written to one CANOE asc output file each.
// CAN #3 and CAN #4 are supposed to be written together to another CANOE asc file.
// All other CAN channels are supposed to be written together to one BLF file.
for (int i = 0; i < nchannels; ++i)
{
    const IChannel &channel = *channels->getChannel(i);
    const ChannelType type = channel.getType();

```

```

const int index = channel.getIndex();

// cout << "ch = " << type << endl;
if (type == CH_CAN)
{
    // Note: channel indices are zero based
    if (index == 0 || index == 1)
    {
        // CAN #1 and #2 in separate files
        // -1 as fileId parameter creates a separate file for this channel
        cs->addChannel(type,
            index,
            CANOE,
            -1,
            channel.getOffset(),
            channel.getMainboardNumber(),
            channel.isMappingActive(),
            channel.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.
        cs->addChannel(type,
            index,
            CANOE,
            10,
            channel.getOffset(),
            channel.getMainboardNumber(),
            channel.isMappingActive(),
            channel.getMappedChannelIndex());
    }
    else
    {
        // All other CAN channels to one BLF file.
        cs->addChannel(type,
            index,
            BLF,
            20,
            channel.getOffset(),
            channel.getMainboardNumber(),
            channel.isMappingActive(),
            channel.getMappedChannelIndex());
    }
}
else
{
    ignoredChannels.insert(type);
    break;
}
}

for (auto c : ignoredChannels)
    cerr << "Ignored channel typ " << c << endl;

ignoredChannels.clear();

cout << "Converting data to \" << path << '\"' << endl;

if (!bpng->convertData(cs, path))
{
    BPNGError err = bpng->getLastError();
    cerr << "online conversion failed: " << err.msg << endl;
    bpng->release();
    return;
}
}

static void offlineConversionTest(const char *offlinefile, const char *path, bool all)
{
    IBPNGClient *bpng = getBPNGClient();

    // register all messages and conversion formats
    bpng->registerAll();

    // set offline data
    bpng->setOfflineData(offlinefile);
}

```

```

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

// Ensure the output directory exists
ret = mkdirPath(path);
if (ret != 0 && errno != EEXIST)
{
    cout << "Failed to create output directory" << endl;
    bpng->release();
    return;
}

IConversionSet *cs = bpng->createNewConversionSet();

const uint64_t us = 1000000U;
time_t startTime, endTime;
tm starttm;
tm endtm;

if (all)
{
    // convert all
    startTime = 0;
    endTime = time(nullptr);

    gmtime_r(&startTime, &starttm);
    gmtime_r(&endTime, &endtm);
}
else
{
    // convert specific time span (adapt this to your needs)
    starttm.tm_sec = 0;
    starttm.tm_min = 0;
    starttm.tm_hour = 0;
    starttm.tm_mday = 8;
    starttm.tm_mon = 1 - 1;
    starttm.tm_year = 2020 - 1900;
    starttm.tm_wday = 0;
    starttm.tm_yday = 0;
    starttm.tm_isdst = 0;

    endtm.tm_sec = 59;
    endtm.tm_min = 59;
    endtm.tm_hour = 23;
    endtm.tm_mday = 9;
    endtm.tm_mon = 12 - 1;
    endtm.tm_year = 2020 - 1900;
    endtm.tm_wday = 0;
    endtm.tm_yday = 0;
    endtm.tm_isdst = 1;

    startTime = mktime(&starttm);
    endTime = mktime(&endtm);
}

cout << "select time span: " << std::put_time(&starttm, "%F %T") << "..." << std::put_time(&endtm, "%F
    %T %Z") << endl;
cs->addTimeSpan(startTime * us, endTime * us);

const IChannelList *channels = bpng->getLoggerChannels();

cout << "channels = " << channels << endl;
if (!channels)
{
    cerr << "no logger channels found" << endl;
    return;
}

const auto nchannels = channels->getSize();

cout << "#channels = " << nchannels << endl;

set<ChannelType> ignoredChannels;

```



```

// With IConversionSet::addChannel() we have to add those channels to the conversion set that we want
// to convert.
// The required format is passed as third argument. Channels that should be written to the same output
// file
// must be added to IConversionSet with the same 'fileId' argument (and of course same formatId).

// In this sample we only want to convert CAN channels according to this:
// CAN #1 and CAN #2 are supposed to be written to one CANOE asc output file each.
// CAN #3 and CAN #4 are supposed to be written together to another CANOE asc file.
// All other CAN channels are supposed to be written together to one BLF file.
for (int i = 0; i < nchannels; ++i)
{
    const IChannel &channel = *channels->getChannel(i);
    const ChannelType type = channel.getType();
    const int index = channel.getIndex();

    // cout << "ch = " << type << endl;
    if (type == CH_CAN)
    {
        // Note: channel indices are zero based
        if (index == 0 || index == 1)
        {
            // CAN #1 and #2 in separate files
            // -1 as fileId parameter creates a separate file for this channel
            cs->addChannel(type,
                index,
                CANOE,
                -1,
                channel.getOffset(),
                channel.getMainboardNumber(),
                channel.isMappingActive(),
                channel.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.
            cs->addChannel(type,
                index,
                CANOE,
                10,
                channel.getOffset(),
                channel.getMainboardNumber(),
                channel.isMappingActive(),
                channel.getMappedChannelIndex());
        }
        else
        {
            // All other CAN channels to one BLF file.
            cs->addChannel(type,
                index,
                BLF,
                20,
                channel.getOffset(),
                channel.getMainboardNumber(),
                channel.isMappingActive(),
                channel.getMappedChannelIndex());
        }
    }
    else
    {
        ignoredChannels.insert(type);
        break;
    }
}

for (auto c : ignoredChannels)
    cerr << "Ignored channel typ " << c << endl;

ignoredChannels.clear();

cout << "Converting data to \" << path << '\"' << endl;

if (!bpng->convertData(cs, path))
{
    BPNGError err = bpng->getLastError();
    cerr << "offline conversion failed: " << err.msg << endl;
}

```

```

        bpng->release();
        return;
    }
}

static void eraseTest(TSLClusterImpl tsl, bool all = true)
{
    IBPNGClient *bpng = getBPNGClient();
    if (!connectInitLogger(__func__, bpng, tsl))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." <<
        endl;
        return;
    }

    if (all)
    {
        cout << "delete all data... ";
        if (bpng->deleteAllData())
        {
            cout << "OK" << endl;
        }
        else
        {
            BPNGError err = bpng->getLastError();
            cerr << "deletion failed: " << err.msg << endl;
            return;
        }
    }
    else
    {
        cout << "get events..." << endl;
        IRdbEventList *list = bpng->getEventList();
        RdbEventList eventList(list);

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

        // download selected time span
        // always from startup to shutdown event
        const uint64_t us = 1000000U;
        time_t startTime, endTime;
        tm starttm;
        tm endtm;

        // start time
        starttm.tm_sec = 0;
        starttm.tm_min = 0;
        starttm.tm_hour = 0;
        starttm.tm_mday = 20;
        starttm.tm_mon = 3 - 1;
        starttm.tm_year = 2020 - 1900;
        starttm.tm_wday = 0;
        starttm.tm_yday = 0;
        starttm.tm_isdst = 0;

        // end time
        endtm.tm_sec = 59;
        endtm.tm_min = 59;
        endtm.tm_hour = 23;
        endtm.tm_mday = 22;
        endtm.tm_mon = 3 - 1;
        endtm.tm_year = 2020 - 1900;
        endtm.tm_wday = 0;
        endtm.tm_yday = 0;
        endtm.tm_isdst = 0;

        startTime = timegm(&starttm) * us;
        endTime = timegm(&endtm) * us;

        cout << "select time span: " << std::put_time(&starttm, "%F %T") << "..." << std::put_time(&endtm, "
        %F %T %Z") << endl;

        vector<size_t> startUpIds;
        // search for startup at starttime and endtime
        for (size_t i = 0; i < eventList.size(); i++)

```

```

    {
        if (eventList[i].type == STARTUP)
        {
            //cout << "event ID: " << eventList[i].uniqueID << " timestamp: "
            unixtimeToLocaltime(eventList[i].timeStamp) << endl;
            if ((eventList[i].timeStamp >= startTime && eventList[i].timeStamp <= endTime)
                || (i > 0 && eventList[i-1].timeStamp <= startTime && eventList[i].timeStamp >=
startTime)
                || (i + 1 < eventList.size() && eventList[i+1].timeStamp >= endTime && eventList[i].
timeStamp <= endTime))
            {
                startUpIds.push_back(eventList[i].uniqueID);
                cout << "event ID: " << eventList[i].uniqueID << " timestamp: "
                    << unixtimeToLocaltime(eventList[i].timeStamp) << endl;
            }
        }
    }

    cout << "number of StartUpIds: " << startUpIds.size() << endl;

    int eraseResult = bpng->deleteSectionsByStartUpIds(startUpIds.size(), &
startUpIds[0]);
    if (eraseResult == 0)
    {
        BPNGError err = bpng->getLastError();
        cerr << "erase failed: " << err.msg << endl;
    }
    else if (eraseResult < 0)
    {
        cerr << "erase failed: user aborted erase!" << endl;
    }
    else
    {
        cout << "erase successfull finished" << endl;
    }
}

bpng->disconnectLogger();
}

static void restartDeviceTest(TSLClusterImpl tsl, bool waitForRestart)
{
    IBPNGClient *bpng = getBPNGClient();
    if (!connectInitLogger(__func__, bpng, tsl))
    {
        cerr << "Failed to initialize logger connection to " << tsl.getTSLName() << ", abort." <<
endl;
        return;
    }

    cout << "restart device(s)... ";
    if (bpng->restartDevice(waitForRestart))
    {
        cout << "OK" << endl;
    }
    else
    {
        BPNGError err = bpng->getLastError();
        cerr << "restart failed: " << err.msg << endl;
        return;
    }

    bpng->disconnectLogger();
}

static void help(const char *prog)
{
    cout << "Usage: " << prog << " <options>\n\n"
        << "Test client lib, where <options> are\n\n";

    cout << "    " << "--all, -a" << "                " << "Download, erase or convert all data, not
just a limited set" << endl;
    cout << "    " << "--config=<TYPE>, -c<TYPE>" << "    " << "Get config, set config, set to default
and restore it.\n" <<
        "                                TYPE={get, default}" << endl << endl;
    cout << "    " << "--download, -d" << "                " << "Download data from loggers" << endl;
    cout << "    " << "--erase, -e" << "                " << "Erase data on loggers" << endl;
    cout << "    " << "--ipAddress[=<IP>], -i[<IP>]" << "    " << "IP address of target (default=" <<
DEFAULT_TARGET_IP << ")" << endl;
    cout << "    " << "--tsl=<NAME>, -t<NAME>" << "        " << "TSL name of target." << endl;
}

```

```

cout << "      " << "--path[=<PATH>], -p[<PATH>]" << "      " << "Path where to to store e.g. download
data, \n" <<
"                                     default is the current directory, for configs this is a file
path" << endl << endl;

cout << "      " << "--gettime, -g" << "      " << "Get time on loggers" << endl;
cout << "      " << "--settime, -s" << "      " << "Set time on loggers" << endl << endl;

cout << "      " << "--offline-conversion=<FILE>, -o<FILE>" << "      " << "Convert offline data"
<< endl << endl;
cout << "      " << "--online-conversion=<FILE>, -O" << "      " << "Convert online data" <
< endl << endl;

cout << "      " << "--restart[=<waitForRestart>], -r<waitForRestart>" << "      " << "Restarts the loggers
(default='0'(false))" << endl;
cout << "      " << "--bugreport[=<mode>], -b<mode>" << "      " << "Download Bugreport (default='full')" <<
endl;

cout << "      " << "--help, -h" << "      " << "This screen." << endl;
cout << "      " << "--verbose, -v" << "      " << "Increase verbosity. Can be specified more
than once." << endl;
}

```

```

int main(int argc, char *argv[])
{
    int c, option_index = 0;
    unsigned int verbose = 0;

    const char * const prog = progname(argv[0]);

    string ip = DEFAULT_TARGET_IP;
    const char *path = ".";
    const char *offlinePath = nullptr;
    string sslName;

    bool all = false;
    bool settime = false;
    bool gettime = false;
    vector<ConfigMode> config;
    bool download = false;
    bool erase = false;
    bool ssl = false;
    bool onlineConversion = false;
    bool doNothing = false;
    bool restart = false;
    bool waitForRestart = false;
    bool bugreport = false;
    BPNGBugreportMode brMode = BR_FULL_WO_TRACES;

    static const struct option long_options[] =
    {
        {"help", no_argument, nullptr, 'h'},
        {"verbose", no_argument, nullptr, 'v'},

        {"all", no_argument, nullptr, 'a'},
        {"config", required_argument, nullptr, 'c'},
        {"download", no_argument, nullptr, 'd'},
        {"erase", no_argument, nullptr, 'e'},
        {"ipAddress", optional_argument, nullptr, 'i'},
        {"ssl", required_argument, nullptr, 't'},
        {"path", optional_argument, nullptr, 'p'},

        {"settime", no_argument, nullptr, 's'},
        {"gettime", no_argument, nullptr, 'g'},

        {"offline-conversion", required_argument, nullptr, 'o'},
        {"online-conversion", no_argument, nullptr, 'O'},

        {"restart", optional_argument, nullptr, 'r'},
        {"bugreport", optional_argument, nullptr, 'b'},

        { nullptr } // Always last entry. Needed to finish parameter list.
    };

    while ((c = getopt_long(argc, argv, "ac:deghi:oOp:svt:r:b:", long_options, &option_index)) != -1)
    {
        switch (c)
        {
            case 'a':

```

```

        all = true;
        break;

    case 'c':
        if (optarg)
            config.push_back(toConfig(optarg));
        break;

    case 'd':
        download = true;
        break;

    case 'e':
        erase = true;
        break;

    case 'i':
        if (optarg)
            ip = optarg;
        break;

    case 't':
        tsl = true;
        if (optarg)
            tslName = optarg;
        break;

    case 'p':
        if (optarg)
            path = optarg;
        break;

    case 's':
        settime = true;
        break;

    case 'g':
        gettime = true;
        break;

    case 'o':
        offlinePath = optarg;
        break;

    case 'O':
        onlineConversion = true;
        break;

    case 'r':
        restart = true;
        if (optarg)
            waitForRestart = atoi(optarg);
        break;

    case 'b':
        bugreport = true;
        if (optarg)
            brMode = toBugreport(optarg);
        break;

    case 'v':
        verbose++;
        break;

    case 'h':
        help(prog);
        doNothing = true;
        break;

    default:
        // already handled by lib
        doNothing = true;
        break;
    }
}

if (offlinePath)
{
    if (verbose)
        writeLogToCout(true);
}

```

```

        offlineConversionTest(offlinePath, path, all);
        doNothing = true;
    }

    if (!doNothing)
    {
        // Get list of all currently available blue Pirat 2 devices
        BPNGLoggerDetector detector(ip, verbose);
        vector<OnlineLoggerInfoWrapper> listOfDevices = detector.getLoggerList(0);
        vector<TSLClusterImpl> listOfTslCluster = detector.getTSLs(listOfDevices);

        // select the device you want to work with
        //      OnlineLoggerInfo device;
        //      for (size_t i = 0; i < listOfDevices.size(); ++i)
        //      {
        //          cout << "[" << i << "]: " << listOfDevices[i] << endl;
        //          if (strcmp(listOfDevices[i].ip, "192.168.0.233") == 0)
        //          {
        //              device = listOfDevices[i];
        //              break;
        //          }
        //      }

        // map for tsl's
        TSLClusterImpl targetTsl;
        for (size_t i = 0; i < listOfTslCluster.size(); ++i)
        {
            TSLClusterImpl tslCluster = listOfTslCluster[i];
            cout << "Found TSL [" << tslCluster.getTSLName() << "]" with devices\n";

            tslCluster.print();

            cout << "\n";

            /* insert target tsl name here*/
            if (tsl && tslCluster.getTSLName() == tslName)
            {
                targetTsl = tslCluster;
            }
        }

        if (!tsl)
        {
            targetTsl = getTslNetwork(ip.c_str(), detector);
        }

        if (targetTsl.begin() != targetTsl.end())
        {
            cout << "targetTsl [" << targetTsl.getTSLName() << "]" with devices\n";
            targetTsl.print();
            cout << "-----\n";

            if (settime)
                setTimeTest(targetTsl);

            if (gettime)
                getTimeTest(targetTsl);

            if (!config.empty())
                configTest(targetTsl, path, config);

            if (download)
                downloadTest(targetTsl, path, all);

            if (onlineConversion)
                onlineConversionTest(targetTsl, path, all);

            if (bugreport)
                bugreportTest(targetTsl, path, brMode);

            if (erase)
            {
                detector.excludeRCTFromTSL(targetTsl);
                eraseTest(targetTsl, all);
            }

            if (restart)
                restartDeviceTest(targetTsl, waitForRestart);
        }
        else
    }

```

```
        {
            cout << "No target set!" << endl;
        }
    }
    return EXIT_SUCCESS;
}
```





## Chapter 2

# Deprecated List

### Member [DEV\\_BP2](#)

For BLUEPIRAT 2 devices use type *DEV\_BP2\_V1X*, for new BLUEPIRAT 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 . . . . .	29
DataSpan . . . . .	37
IBPNGClient . . . . .	37
IBPNGClientListener . . . . .	59
BPNGLoggerDetector . . . . .	29
IChannel . . . . .	66
IChannelList . . . . .	66
IClientProperties . . . . .	67
IConversionSet . . . . .	75
IFalseMeasureSignal . . . . .	77
IFalseMeasureSignalList . . . . .	78
IFormatInfo . . . . .	78
IFormatList . . . . .	79
IRdbEvent . . . . .	79
IRdbEventList . . . . .	80
IRdbTraceBlock . . . . .	81
IRdbTraceBlockList . . . . .	81
ITesttoolsChannel . . . . .	81
ITesttoolsChannelList . . . . .	82
LogInData . . . . .	83
MemoryFillLevel . . . . .	83
OnlineLoggerInfo . . . . .	84
OnlineLoggerInfoWrapper . . . . .	86
OnlineLoggerInfoStringPair . . . . .	86
RdbEvent2 . . . . .	88
RdbTraceBlock2 . . . . .	89
vector	
RdbEventList . . . . .	88
RdbTraceBlockList . . . . .	90
TSLCluster . . . . .	91
TSLClusterImpl . . . . .	91



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">BPNGError</a>	
Error struct with error code and optional error message . . . . .	29
<a href="#">BPNGLoggerDetector</a> . . . . .	29
<a href="#">DataSpan</a>	
Structure of a data span . . . . .	37
<a href="#">IBPNGClient</a>	
Interface class for the Telemotive Client Library . . . . .	37
<a href="#">IBPNGClientListener</a> . . . . .	59
<a href="#">IChannel</a>	
Channel interface . . . . .	66
<a href="#">IChannelList</a>	
Channel list interface . . . . .	66
<a href="#">IClientProperties</a>	
The <a href="#">IClientProperties</a> interface replaces the deprecated <i>ClientProperties</i> struct	67
<a href="#">IConversionSet</a>	
A conversion set stores all conversion relevant settings . . . . .	75
<a href="#">IFalseMeasureSignal</a>	
False measure signal interface . . . . .	77
<a href="#">IFalseMeasureSignalList</a>	
False measure signal list interface . . . . .	78
<a href="#">IFormatInfo</a>	
FormatInfo interface . . . . .	78
<a href="#">IFormatList</a>	
Format list interface . . . . .	79
<a href="#">IRdbEvent</a>	
Interface to an RDB event . . . . .	79
<a href="#">IRdbEventList</a>	
Interface to a list of rdb events . . . . .	80
<a href="#">IRdbTraceBlock</a> . . . . .	81
<a href="#">IRdbTraceBlockList</a> . . . . .	81
<a href="#">ITesttoolsChannel</a>	
Channel interface . . . . .	81
<a href="#">ITesttoolsChannelList</a>	
TesttoolsChannel list interface . . . . .	82

<a href="#">LoginData</a>	
Structure for login	83
<a href="#">MemoryFillLevel</a>	
Stores memory fill level of a device	83
<a href="#">OnlineLoggerInfo</a>	
Struct with information about a logger found in LAN/WLAN used to notify <a href="#">IBP-NGClientListener</a> about detected/disappeared devices	84
<a href="#">OnlineLoggerInfoStringPair</a>	
Helper object for configuration, license update or firmwareupdate: a key value pair for assigning a configuration, licensefile, etc. to a device	86
<a href="#">OnlineLoggerInfoWrapper</a>	
Wrapper around brain dead <a href="#">OnlineLoggerInfo</a>	86
<a href="#">RdbEvent2</a>	
Implementation class for a wrapper of <a href="#">IRdbEvent</a> using STL classes	88
<a href="#">RdbEventList</a>	
Implementation class for a wrapper of <a href="#">IRdbEventList</a> using STL classes	88
<a href="#">RdbTraceBlock2</a>	89
<a href="#">RdbTraceBlockList</a>	90
<a href="#">TSLCluster</a>	
Representation of a chain of Telemotive devices combined via Telemotive System Link (TSL)	91
<a href="#">TSLClusterImpl</a>	91

## Chapter 5

# File Index

### 5.1 File List

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

<a href="#">BPNGDefines.h</a>	Defines for Telemotive Client Library . . . . .	95
<a href="#">BPNGLoggerDetector.hh</a>	<a href="#">IBPNGClientListener</a> wrapper . . . . .	106
<a href="#">clientUtil.hh</a>	Helper functions . . . . .	107
<a href="#">IBPNGClient.h</a>	Interface class for the BPNGClient DLL . . . . .	110
<a href="#">IBPNGClientListener.h</a>	Interface class for the BPNGClient listener . . . . .	113
<a href="#">IClientProperties.h</a>	Interface for client properties . . . . .	113
<a href="#">OnlineLoggerInfoWrapper.hh</a>	C++ wrapper around brain dead <a href="#">OnlineLoggerInfo</a> . . . . .	114
<a href="#">RdbDefines.h</a>	Public interfaces for Telemotive Reference Database access . . . . .	114
<a href="#">RdbEventList.hh</a>	<a href="#">IRdbEvent</a> wrapper . . . . .	116
<a href="#">RdbTraceBlockList.hh</a>	<a href="#">IRdbTraceBlock</a> wrapper . . . . .	116
<a href="#">TSLClusterImpl.hh</a>	<a href="#">TSLCluster</a> wrapper . . . . .	117





## 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

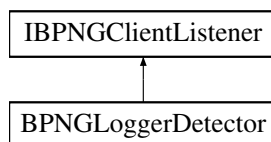
Error struct with error code and optional error message.

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

- [BPNGDefines.h](#)

### 6.2 BPNGLoggerDetector Class Reference

Inheritance diagram for BPNGLoggerDetector:



#### Public Member Functions

- `template<typename IP >`  
[BPNGLoggerDetector](#) (`IP &&_ip`, `unsigned int _debug`)  
*CTOR.*
- `virtual void` [onBPNGDeviceDetected](#) ([OnlineLoggerInfo](#) \*`info`)

- Called to notify a detected logger in network.*

  - virtual void **onBPNGDeviceDisappeared** ([OnlineLoggerInfo](#) \*info)
- Called to notify a disappeared logger.*

  - virtual void **onBPNGDeviceStateChange** ([OnlineLoggerInfo](#) \*info)
- Called to notify a logger's state change.*

  - virtual int **onProgressDataDownload** (int percentCompleted)
- Called to indicate the current progress of a data transfer.*

  - virtual int **onProgressDataDownload** (int percentCompleted, uint64\_t downloadedSize, uint64\_t totalSize)
- Called to indicate the current progress of a data transfer.*

  - virtual int **onProgressConversion** (int percentCompleted, const char \*status)
- Called to indicate the current progress of file conversion.*

  - virtual int **onProgressDeletion** (int percentCompleted)
- Called to indicate the current progress of file deletion.*

  - virtual int **onTargetPathTooLong** (char \*newTarget, int maxSize)
- Called on a too long target directory.*

  - virtual int **getOverwritingPermission** (const char \*filePath)
- Called on existing output trace files.*

  - virtual void **onFirmwareUpdateProgress** (int percentage, int step, int substep, const char \*desc)
- Called on firmware update progress.*

  - virtual void **onFirmwareUpdateError** (int errorId)
- Called in case of not enough free disk space.*

  - virtual int **onCriticalDiskSpace** (uint64\_t freeSpace, uint64\_t neededSpace, const char \*drive, const char \*msg)
- Called on existing output trace files.*

  - virtual void **onLoggerConnected** (bool isConnected)
- Called to send additional information of the current process to the calling app.*

  - virtual void **onStatusMessage** (const char \*statusMsg)
- Called to send additional information of the current process to the calling app.*

  - virtual int **onGetLogReportProgress** (int percentage, const char \*desc)
- Called to inform about a warning.*

  - virtual void **onWarning** ([BPNGWarningCode](#) warningCode, const char \*warnMsg)
- Called to inform about a warning.*

  - virtual void **onInvalidPwConfigFound** (const char \*)
- Notifies the listeners before the download starts about the total amount of bytes to be downloaded.*

  - virtual void **onDownloadStart** (int64\_t totalAmountOfBytes)
- Notifies the listeners before the download starts about the total amount of bytes to be downloaded.*

  - virtual void **onLogInDataFailed** ()
- Notifies the listeners before the download starts about the total amount of bytes to be downloaded.*

  - virtual void **onResetLogInDataFailed** ()
- Notifies the listeners before the download starts about the total amount of bytes to be downloaded.*

  - virtual void **onFuncAccessDenied** ()
- Notifies the listeners before the conversion starts about the total amount of bytes to be converted.*

  - virtual void **onConversionStart** (int64\_t totalAmountOfBytes)
- Notifies the listeners before the conversion starts about the total amount of bytes to be converted.*

  - virtual int **onDataRecoverProgress** (const char \*test1, int test2)
- Called to send additional information of the current data recovery progress.*

  - virtual const char \* **onLogInDataRequired** (unsigned int test1)
- Called if invalid pw file found on device.*

  - virtual void **onInvalidPwConfigFound** (unsigned int)
- Called if invalid pw file found on device.*

  - virtual const char \* **onExtractionPasswordRequired** (unsigned int)
- Called periodically on live download to query whether the permanent download should be finished.*

  - virtual bool **isTerminateLiveDownloadRequest** ()
- Called periodically on live download to query whether the permanent download should be finished.*

  - [OnlineLoggerInfo](#) **getLoggerInfoForIP** (const char \*ip) const

- const std::string & **getIP** () const
- std::vector< [OnlineLoggerInfoWrapper](#) > **getLoggerList** (unsigned searchTimeOut)
- std::vector< [TSLClusterImpl](#) > **getTSLs** (const std::vector< [OnlineLoggerInfoWrapper](#) > &tslChain)
- void **excludeRCTFromTSL** ([TSLClusterImpl](#) &loggersInNetwork)

## 6.2.1 Member Function Documentation

### excludeRCTFromTSL()

```
void BPNGLoggerDetector::excludeRCTFromTSL (
    TSLClusterImpl & loggersInNetwork )
```

Remove RCT from TSL for spezific functions like delete data.

#### Parameters

<i>loggersInNetwork</i>	TSL Chain to check for rct devices
-------------------------	------------------------------------

### getLoggerList()

```
std::vector<OnlineLoggerInfoWrapper> 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

### getOverwritingPermission()

```
virtual int BPNGLoggerDetector::getOverwritingPermission (
    const char * filePath ) [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](#).

### getTSLs()

```
std::vector<TSLClusterImpl> BPNGLoggerDetector::getTSLs (
    const std::vector< OnlineLoggerInfoWrapper > & tslChain )
```

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 <a href="#">getLoggerList(unsigned searchTimeout)</a>
-------------------------	--

#### Returns

vector of [TSLCluster](#)

#### onBPNGDeviceDetected()

```
virtual void 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](#).

#### onBPNGDeviceDisappeared()

```
virtual void 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](#).

#### onBPNGDeviceStateChange()

```
virtual void 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](#).

#### onConversionStart()

```
virtual void BPNGLoggerDetector::onConversionStart (
    int64_t totalAmountOfBytes ) [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](#).

### onCriticalDiskSpace()

```
virtual int BPNGLoggerDetector::onCriticalDiskSpace (
    uint64_t freeSpace,
    uint64_t neededSpace,
    const char * drive,
    const char * msg ) [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](#).

### onDataRecoverProgress()

```
virtual int 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](#).

### onDownloadStart()

```
virtual void BPNGLoggerDetector::onDownloadStart (
    int64_t totalAmountOfBytes ) [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](#).

### onExtractionPasswordRequired()

```
virtual const char* 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](#).

### onGetLogReportProgress()

```
virtual int BPNGLoggerDetector::onGetLogReportProgress (
    int percentage,
    const char * desc ) [virtual]
```

Called on creation of log report

#### Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

### onInvalidPwConfigFound()

```
virtual void 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](#).

### onProgressConversion()

```
virtual int BPNGLoggerDetector::onProgressConversion (
    int percentCompleted,
    const char * status ) [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](#).

**onProgressDataDownload()** [1/2]

```
virtual int BPNGLoggerDetector::onProgressDataDownload (
    int percentCompleted ) [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](#).

**onProgressDataDownload()** [2/2]

```
virtual int BPNGLoggerDetector::onProgressDataDownload (
    int percentCompleted,
    uint64_t downloadedSize,
    uint64_t totalSize ) [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.
-------------------------	---

**Parameters**

<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](#).

**onProgressDeletion()**

```
virtual int BPNGLoggerDetector::onProgressDeletion (
    int percentCompleted ) [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](#).

**onStatusMessage()**

```
virtual void BPNGLoggerDetector::onStatusMessage (
    const char * statusMsg ) [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](#).

**onTargetPathTooLong()**

```
virtual int BPNGLoggerDetector::onTargetPathTooLong (
    char * newTarget,
    int maxSize ) [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](#).

### onWarning()

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

structure of a data span

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

### 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**

#### 6.3.1 Detailed Description

structure of a data span

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 [getInstanceld](#) ()=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](#) (PwPrivilegesFuncId 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 \*startuplds)=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 CCPXCPConfiguration.xml and create CCPXCPSequence.xml.*
- virtual int WINAPI [createCCPXCPDbcFiles](#) (const char \*dbcDir, const char \*xsdDir, const char \*xmlDir)=0  
*Parse CCPXCPMeasurement.xml and CCPXCPConfiguration.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 BLUEPIRAT Client library. To get access to any Telemotive data logger except "BLUEPIRAT 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

**activateGatewayLoggerDetection()**

```
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.

**assignDatabaseFile()**

```
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\_MIL: 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

**assignDBCFile()**

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

**clearDBCFileAssignments()**

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

Remove all DBC file assignments.

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

**will be removed with next vesion!**

**connectLogger()**

```
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 <a href="#">OnlineLoggerInfo</a> devices
<i>devices</i>	pointer to first <a href="#">OnlineLoggerInfo</a>

#### Returns

0 on failure, 1 on success

### convertData()

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

#### Returns

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

### createNewConversionSet()

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

Returns the pointer to a new conversion set.

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

**createTestReport()**

```
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 <a href="#">IConversionSet</a>
<i>target</i>	target directory for the test reports.

**Returns**

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

**deleteAllData()**

```
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.

**deleteSectionsByStartUpIds()**

```
virtual int WINAPI IBPNGClient::deleteSectionsByStartUpIds (
    uint16_t numStartupIds,
    uint64_t * startupIds ) [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 startupIds 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 BLUEPIRAT Collect devices (BPNGDeviceType DEV\_TRACE\_COLL\_DS).

**Parameters**

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



## Returns

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

**downloadBugReport()**

```
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.

**downloadDataSpans()**

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

#### Returns

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

### enableClientLogOutput()

```
virtual void WINAPI IBPNGClient::enableClientLogOutput ( ) [pure virtual]
    enable ClientLogUtil output
    This function enable the output of class ClientLogUtil.
```

### filterSignals()

```
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>pathToFilterSetting</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

### filterSignalsFromOfflineData()

```
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>pathToOfflineDataSet</i>	path to the offline data set
<i>pathToFilterSetting</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

#### flashDeviceLED()

```
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.

#### getAvailableFormats()

```
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](#)

#### getClientProperties()

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

See also

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

#### getConfig()

```
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
-------------	--

### getConfigPath()

```
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:\...\BLUEPIRAT\PNGINST25452\BPTSL\_10.64.76.171\_2\BP2Img\_10.64.76.171\_3\_PID5452**

#### Returns

Path to the folder containing the extracted config archive.

### getConversionError()

```
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.

### getDeviceName()

```
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\(\)](#)

**getDownloadError()**

```
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.

**getEventList()**

```
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](#)

**getFalseMeasureSignals()**

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

Return pointer to a false measure signal list interface.

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

See also

[IFalseMeasureSignal](#)

**getLastError()**

```
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](#)

**getLicenses()**

```
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

**getLoggerChannels()**

```
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](#)

**getMemoryFillLevel()**

```
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

**getNumConversionErrors()**

```
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\(\)](#)

**getNumDownloadErrors()**

```
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\(\)](#)

**getPwdFile()**

```
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

**getReferenceDataBasePath()**

```
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\(\)](#)

**getTraceBlockList()**

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

Get list of all trace blocks from the RDB.

## Returns

Pointer to a [IRdbEventList](#)

**getVersions()**

```
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

**initialize()**

```
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

**isPasswordProtectionSupported()**

```
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

**keepLoggerAlive()**

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

Call this to keep logger alive.

The BLUEPIRAT 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\(\)](#)

**reconfigLogger()**

```
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 <a href="#">OnlineLoggerInfoStringPair</a> (should be equal to the number of devices on TSL)
<i>loggerToConfigPathPairs</i>	Pointer to first <a href="#">OnlineLoggerInfoStringPair</a>
<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

#### release()

```
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!

#### removeAllLicenses()

```
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

#### restartDevice()

```
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

**scanNetworkForLogger()**

```
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.

**setClientProperties()**

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

## Parameters

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

## See also

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

**setDefaultConfig()**

```
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

**setDevice()**

```
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](#)

### setInfoEvent()

```
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

### setMarker()

```
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

### setOfflineData()

```
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\(\)](#).

### setPwdFile()

```
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

**setTime()**

```
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

**setTSLCluster()**

```
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();
```

**shutdownDevice()**

```
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

**startLiveDownload()**

```
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

**synchronizeRdb()**

```
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.

**updateFirmware()**

```
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 than the component's version inside the firmware packet will be updated.

**Parameters**

<i>loggerToFirmwareUpdatePacketPair</i>	A pair with key= <code>OnlineLoggerInfo</code> , 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

**updateLicenses()**

```
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= <a href="#">OnlineLoggerInfo</a> , the device to be updated and value=local path to licsene file
------------------------------	---

#### Returns

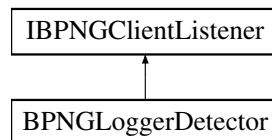
0 on failure, 1 on success

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

#### **getOverwritingPermission()**

```
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](#).

#### **onBPNGDeviceDetected()**

```
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](#).



**onBPNGDeviceDisappeared()**

```
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](#).

**onBPNGDeviceStateChange()**

```
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](#).

**onConversionStart()**

```
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](#).

**onCriticalDiskSpace()**

```
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](#).

**onDataRecoverProgress()**

```
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](#).

**onDownloadStart()**

```
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](#).

**onExtractionPasswordRequired()**

```
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](#).

**onGetLogReportProgress()**

```
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](#).

### onInvalidPwConfigFound()

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

Called if invalid pw file found on device.

An error may occur 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](#).

### onLogInDataRequired()

```
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.

### onProgressConversion()

```
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
-------------------------	--

**Parameters**

<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](#).

**onProgressDataDownload()** [1/2]

```
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](#).

**onProgressDataDownload()** [2/2]

```
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](#).

**onProgressDeletion()**

```
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](#).

**onStatusMessage()**

```
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](#).

**onTargetPathTooLong()**

```
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](#).

**onWarning()**

```
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 mainboard number of device the channel belongs to.*
- 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

#### addAnalogPortSettings()

```
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

#### addDigitalPortSettings()

```
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

**setCANPseudoMsgTimeStampProperties()**

```
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>writeTimeStampMsg</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

**setCANPseudoMsgTriggerProperties()**

```
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
----------------------------	--

## Parameters

<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)

**setCommonProperties()**

```
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.

**Parameters**

<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 <a href="#">IConversionSet</a> . 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)

**setFlexRayPseudoMsgProperties()**

```
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

**setMOSTPseudoMsgProperties()**

```
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

### addChannel()

```
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 strings. Can be read out from <a href="#">file format identifier</a>
<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 <a href="#">IChannel</a>
<i>mbnr</i>	Only needed for TSL, default -1. The mainboardnumber of the channels source device. Can be read out from <a href="#">IChannel</a>
<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 <a href="#">IChannel</a>
<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 <a href="#">IChannel</a>

### addRdbIdRange()

```
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

### addTimeSpan()

```
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

### loadConversionFilters()

```
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

#### getComment()

```
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.

#### getTimeZone()

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

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

#### getUniqueId()

```
virtual uint64_t WINAPI IRdbEvent::getUniqueId ( ) 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:

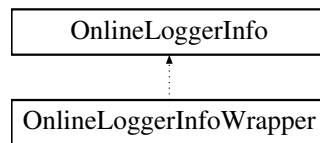
- [BPNGDefines.h](#)

## 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>
```

Inheritance diagram for OnlineLoggerInfo:



### 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

#### deviceType

`uint8_t OnlineLoggerInfo::deviceType`  
*Device type,.*

See also

[BPNGDeviceType](#)

**loggerStatus**

```
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:

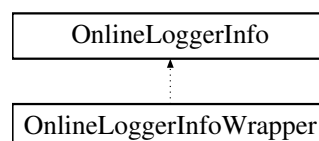
- [BPNGDefines.h](#)

**6.24 OnlineLoggerInfoWrapper Class Reference**

Wrapper around brain dead [OnlineLoggerInfo](#).

```
#include <OnlineLoggerInfoWrapper.hh>
```

Inheritance diagram for OnlineLoggerInfoWrapper:



## Public Member Functions

- [OnlineLoggerInfoWrapper](#) ()  
*Default CTOR.*
- [OnlineLoggerInfoWrapper](#) (const char \*\_ip, const char \*\_name)  
*CTOR taking an IP and a name.*
- [OnlineLoggerInfoWrapper](#) (const [OnlineLoggerInfo](#) &info)  
*CTOR from the "original".*
- [OnlineLoggerInfoWrapper](#) (const [OnlineLoggerInfoWrapper](#) &info)  
*Copy CTOR.*
- [OnlineLoggerInfoWrapper](#) ([OnlineLoggerInfoWrapper](#) &&info)  
*Move CTOR.*
- [OnlineLoggerInfoWrapper](#) & operator= (const [OnlineLoggerInfo](#) &info)  
*Copy assignement from the "original".*
- [OnlineLoggerInfoWrapper](#) & operator= (const [OnlineLoggerInfoWrapper](#) &info)  
*Copy assignement.*
- [OnlineLoggerInfoWrapper](#) & operator= ([OnlineLoggerInfoWrapper](#) &&info)  
*Move assignement.*
- [~OnlineLoggerInfoWrapper](#) ()  
*DTOR.*
- const [OnlineLoggerInfo](#) & getInfo () const  
*Get [OnlineLoggerInfo](#).*
- void setIP (const char \*\_ip)  
*Set logger's IP.*
- const char \* getIP () const  
*Get logger's IP.*
- void setName (const char \*\_ip)  
*Set logger's name.*
- const char \* getName () const  
*Get logger's name.*
- void setMbnr (const char \*\_ip)  
*Set logger's mainboard number.*
- const char \* getMbnr () const  
*Get logger's mainboard number.*
- int32\_t getTslNetworkId () const  
*Get logger's TSL network ID.*

## Friends

- std::ostream & operator<< (std::ostream &os, const [OnlineLoggerInfoWrapper](#) &oli)  
*Output operator.*

### 6.24.1 Detailed Description

Wrapper around brain dead [OnlineLoggerInfo](#).

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

- [OnlineLoggerInfoWrapper.hh](#)

## 6.25 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**

#### 6.25.1 Detailed Description

Implementation class for a wrapper of [IRdbEvent](#) 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 [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:

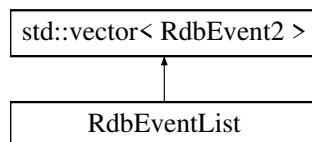
- [RdbEventList.hh](#)

## 6.26 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)

### 6.26.1 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](#)

## 6.27 RdbTraceBlock2 Struct Reference

```
#include <RdbTraceBlockList.hh>
```

### Public Member Functions

- **RdbTraceBlock2** (const [IRdbTraceBlock](#) \*rdbBlock)

### Public Attributes

- uint64\_t **m\_DataBaseEntryId**
- std::string **m\_LoggerModuleName**
- std::string **m\_FilePath**
- std::string **m\_FileName**
- uint64\_t **m\_DataFileSize**
- uint64\_t **m\_DataSize**
- uint64\_t **m\_DataStartTimeUTC**  
*start time of trace block in us*
- uint64\_t **m\_DataEndTimeUTC**  
*end time of trace block in us*
- std::string **m\_DataStartGPS**
- std::string **m\_DataEndGPS**
- uint64\_t **m\_BlockNr**
- std::string **m\_TimeZone**
- std::string **m\_CfgBackupFile**
- std::string **m\_CAN\_CANextData**
- std::string **m\_MOST25Data**
- std::string **m\_SerialData**
- std::string **m\_EthernetData**
- std::string **m\_FlexRayData**
- std::string **m\_LINData**
- std::string **m\_ApixData**
- std::string **m\_MOST150Data**
- std::string **m\_CameraData**
- std::string **m\_AnalogData**

- `std::string m_GpioData`
- `std::string m_AudioData`
- `std::string m_CCPXCPData`
- `std::string m_DiagData`
- `std::string m_GPSPData`
- `std::string m_ECLData`
- `std::string m_CLASSData`
- `std::string m_ComplexFilterData`
- `std::string m_TTYData`
- `std::string m_MIIData`
- `std::string m_Comment`

### 6.27.1 Detailed Description

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

See also

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

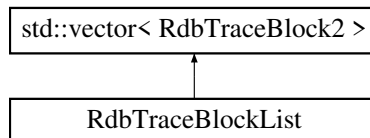
The documentation for this struct was generated from the following file:

- [RdbTraceBlockList.hh](#)

## 6.28 RdbTraceBlockList Class Reference

```
#include <RdbTraceBlockList.hh>
```

Inheritance diagram for [RdbTraceBlockList](#):



### Public Member Functions

- **RdbTraceBlockList** (const [IRdbTraceBlockList](#) \*list)
- void [sortTraceBlocksById](#) ()

*Sort trace blocks by their data base entry id.*

### 6.28.1 Detailed Description

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

See also

[RdbTraceBlock](#), [IRdbTraceBlockList](#), [IRdbTraceBlock](#)

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

- [RdbTraceBlockList.hh](#)

## 6.29 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`

### 6.29.1 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](#)

## 6.30 TSLClusterImpl Class Reference

```
#include <TSLClusterImpl.hh>
```

### Public Types

- enum [ConnectionType](#) {  
[DOWNLOAD](#), [CONVERSION](#), [CONFIG](#), [BUGREPORT](#),  
[FW\\_UPDATE](#) }

*A enumeration of types of which task the connection will be used for.*

### Public Member Functions

- [TSLClusterImpl](#) ()  
*Constructor.*
- [TSLClusterImpl](#) ([OnlineLoggerInfo](#) firstDevice)
- void [addDevice](#) ([OnlineLoggerInfo](#) device)
- void [deleteDevice](#) (size\_t index)
- std::string [getTSLName](#) ()
- void [print](#) ()  
*Stream the TSLCluster to cout.*
- std::vector< [OnlineLoggerInfo](#) >::iterator [begin](#) ()
- std::vector< [OnlineLoggerInfo](#) >::iterator [end](#) ()
- [TSLCluster](#) [getTSLCluster](#) ()
- size\_t [getTSLSize](#) ()  
*get size of TSL chain*

### 6.30.1 Detailed Description

A simple class that represents a Telemotive System Link chain.

### 6.30.2 Member Enumeration Documentation

#### ConnectionType

```
enum TSLClusterImpl::ConnectionType
```

A enumeration of types of which task the connection will be used for.

#### Enumerator

DOWNLOAD	Download tasks.
CONVERSION	Conversion tasks.
CONFIG	Configuration tasks.
BUGREPORT	Create bug report.
FW_UPDATE	make Firmware update

### 6.30.3 Constructor & Destructor Documentation

#### TSLClusterImpl()

```
TSLClusterImpl::TSLClusterImpl (
    OnlineLoggerInfo firstDevice ) [inline]
```

Constructor

#### Parameters

<i>firstDevice</i>	the first device of the chain.
--------------------	--------------------------------

### 6.30.4 Member Function Documentation

#### addDevice()

```
void TSLClusterImpl::addDevice (
    OnlineLoggerInfo device ) [inline]
```

Add a BPNGDevice to the [TSLCluster](#).

#### Parameters

<i>device</i>	the new device
---------------	----------------



**begin()**

```
std::vector<OnlineLoggerInfo>::iterator TSLClusterImpl::begin ( ) [inline]
```

Begin iterator for ranged base for loop

Returns

the begin iterator of internal BPNGDevice vector.

**deleteDevice()**

```
void TSLClusterImpl::deleteDevice (
    size_t index ) [inline]
```

Delete a BPNGDevice from the [TSLCluster](#).

Parameters

<i>index</i>	index of device to delete
--------------	---------------------------

**end()**

```
std::vector<OnlineLoggerInfo>::iterator TSLClusterImpl::end ( ) [inline]
```

End iterator for ranged base for loop

Returns

the end iterator of internal BPNGDevice vector.

**getTSLName()**

```
std::string TSLClusterImpl::getTSLName ( ) [inline]
```

Get the name of TSL chain. All devices in the chain have the same TSL name.

Returns

the TSL name.

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

- [TSLClusterImpl.hh](#)



## 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](#)  
*structure of a data span*

- 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 **BPNG\_CLIENT\_LIB\_VERSION** "5.1.0.8"

## 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_ERRO
= 44, BPNG_NOTHING_TO_DOWNLOAD = 45,
BPNG_INVALID_OFFLINE_SET = 46, BPNG_PARAMETER_MISMATCH = 47, BPNG_FW_VERSION_CHEC
= 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_GE
= 55, BPNG_CCP_XCP_SEQUENCE_GENERATOR_ERROR = 56,
BPNG_INSUFFICIENT_DISK_SPACE = 57, BPNG_FWUPDATE_FAILED = 58, BPNG_INDEX_OUT_OF_RA
= 59, BPNG_READ_CONFIG_BACKUP_ERR = 60,
BPNG_INVALID_RPC_COMMAND = 61, BPNG_INVALID_TSL_CASCDING = 62, BPNG_LOGIN_CANCELE
= 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_INCONSISTEN
= 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_OFFL
= 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_DAT
= 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 FWUpdateErrorCode {
FWUPDATE_ERRORCODE_NO_ERR = 0, FWUPDATE_ERRORCODE_FW_PKT_NAME_EMPTY
= -2, FWUPDATE_ERRORCODE_FW_PKT_MISSING = -3, FWUPDATE_ERRORCODE_NAMED_PIPE_SE
= -4,
FWUPDATE_ERRORCODE_NAMED_PIPE_SERVER_OPEN = -5, FWUPDATE_ERRORCODE_FW_UPDAT
= -6, FWUPDATE_ERRORCODE_MISSING_LINUX_DISTR = -7, FWUPDATE_ERRORCODE_MISSING_LIE
= -8,
FWUPDATE_ERRORCODE_MISSING_TMLIB_FILE = -9, FWUPDATE_ERRORCODE_MISSING_ATOM_FI
= -10, FWUPDATE_ERRORCODE_MISSING_CLIENT_FILE = -11, FWUPDATE_ERRORCODE_MISSING_F
= -12,
FWUPDATE_ERRORCODE_MISSING_FPGAB_FILE = -13, FWUPDATE_ERRORCODE_MISSING_EXTEN
= -14, FWUPDATE_ERRORCODE_MISSING_GBE_FILE = -15, FWUPDATE_ERRORCODE_MISSING_SBC
= -16,
FWUPDATE_ERRORCODE_MISSING_SBC_FLASH_SCRIPT = -17, FWUPDATE_ERRORCODE_MISSING
= -18, FWUPDATE_ERRORCODE_MISSING_RCV_FILE = -19, FWUPDATE_ERRORCODE_MISSING_LIN
= -20,
FWUPDATE_ERRORCODE_UNKNOWN_MB_HW_VERSION = -21, FWUPDATE_ERRORCODE_UNKNOW
= -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, FWUP-
DATE_ERRORCODE_FAILED_UPDATE_APP_LIBS = -27, FWUPDATE_ERRORCODE_FAILED_UPDATE
= -28,
FWUPDATE_ERRORCODE_FAILED_UPDATE_GBEC = -29, FWUPDATE_ERRORCODE_CONV_CFG_ER
= -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_
= -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
}
• enum LanguageID { BPNG_GERMAN, BPNG_ENGLISH }
• 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 }
• 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 }
• 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 }

```

- 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  
}
- enum **DataSpanType** { **DST\_IDSPAN** = 0, **DST\_TIMESPAN** = 1 }  
Types for **DataSpan**.

## Variables

- static const char \*const **UNDEFINED** = "UNDEFINED"  
Following static variables are identifier for currently supported file formats for data conversion.
- static const char \*const **TMASC** = "TMASC"  
Telemotive ASCII format.
- static const char \*const **OP2** = "OP2"  
MOST OP" format.
- static const char \*const **CANOE** = "CANOE"  
CANOE ASCII format.
- static const char \*const **STA** = "STA"  
Serial Trace Analyser format.
- static const char \*const **GNLOG** = "GNLOG"  
GNLog format.
- static const char \*const **TCLOG** = "TCLOG"  
Trace Client format.
- static const char \*const **TCLOG\_TS** = "TCLOG\_TS"  
Trace Client format plus time stamps.
- static const char \*const **RAW\_SERIAL** = "RAW\_SERIAL"  
Raw Serial Format.
- static const char \*const **IMG** = "IMG"  
DataAnalyser IMG format.
- static const char \*const **TMBIN** = "TMBIN"  
Telemotive binary format.
- static const char \*const **CANCORDER** = "CANCORDER"  
CANCOrder format.
- static const char \*const **APN** = "APN"  
APN serial format.
- static const char \*const **ASCHEX** = "ASCHEX"  
ASCII format that writes binary data as hex values.
- static const char \*const **TCPDUMP** = "TCPDUMP"  
TCP dump (\*.pcap) format.
- static const char \*const **BLF** = "BLF"  
Binary Logging Format - Vector Informatik.
- static const char \*const **DLT\_BMW** = "DLT\_BMW"  
BMW specific DLT Format.
- static const char \*const **MDF** = "MDF"  
Measurement data format - Vector Informatik.
- static const char \*const **MDF\_CAN\_SIG** = "MDF\_CAN\_SIG"

- MDF format based on CAN signals.*

  - static const char \*const **MPEG4\_BLOCKS** = "MPEG4\_BLOCKS"

*Camera format seperated blocks.*

  - static const char \*const **MPEG4\_JOINED** = "MPEG4\_JOINED"

*Camera joined videos.*

  - static const char \*const **NMEA** = "NMEA"

*Raw format for GPS data.*

  - static const char \*const **KML** = "KML"

*KML format for GPS data.*

  - static const char \*const **KMZ** = "KMZ"

*KMZ format for GPS data.*

  - static const char \*const **SERIAL\_DEBUG** = "SERIAL\_DEBUG"

*Serial Format with "new Line"-packaging.*

  - static const char \*const **RAW\_ETHERNET** = "RAW\_ETHERNET"

*Raw ethernet format.*

  - static const char \*const **ESO\_TRACE** = "ESO\_TRACE"

*Audi specific ESO Format.*

  - static const char \*const **XTMT** = "XTMT"

*Extended Telemotive Trace File format.*

  - static const char \*const **GPX** = "GPX"

*GPS Exchange format.*

  - static const char \*const **MDF\_CCP\_XCP\_SIG** = "MDF\_CCP\_XCP\_SIG"

*MDF format based on CCP & XCP signals.*

  - static const char \*const **TS** = "TS"

*Raw isochrone format.*

  - static const char \*const **FILTER\_FORMAT** = "FILTER\_FORMAT"

*Filter format= no change in format.*

  - static const char \*const **MDF\_SIG\_LOG\_4\_10** = "MDF\_SIG\_LOG\_4\_10"

*MDF 4.10 format for signal logging.*

  - static const char \*const **MDF\_BUS\_LOG\_4\_10** = "MDF\_BUS\_LOG\_4\_10"

*MDF 4.10 format for bus logging.*

  - static const char \*const **QXDM\_RAW\_SERIAL** = "QXDM\_RAW\_SERIAL"

*Like RAW\_SERIAL but only for tty and with file extension \*.qmdl.*

  - static const char \*const **PCAPNG** = "PCAPNG"

*Wireshark - pcapng (\*.pcapng) format.*

  - static const char \*const **INVALID** = "INVALID"

*Last identifier of file formats for data conversion.*

  - static const char \*const **WULASC** = "WULASC"

*WakeupLine.*

### 7.1.1 Detailed Description

Defines for Telemotive Client Library.

### 7.1.2 Enumeration Type Documentation



**BPNGBugreportMode**

enum [BPNGBugreportMode](#)

Available modes for the creation of a bugreport. 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.

**BPNGDeviceType**

enum [BPNGDeviceType](#)

Enumartion of Telemotives next generation data loggers

**Enumerator**

DEV_BP2	<b>Deprecated</b> For BLUEPIRAT 2 devices use type <i>DEV_BP2_V1X</i> , for new BLUEPIRAT 2 5E devices use <i>DEV_BP2_V2X</i>
DEV_BPMINI	BLUEPIRAT mini devices.
DEV_BP2_V1X	standard BLUEPIRAT 2 device
DEV_BP2_V2X	BLUEPIRAT 2 5E device.
DEV_RC_TOUCH	Remote Control Touch.
DEV_BP_REMOTE	BLUEPIRAT Remote.
DEV_BP_TOUCH	BLUEPIRAT Touch.
DEV_RAPID	BLUEPIRAT Rapid.
DEV_TRACE_COLL	Trace Collector.
DEV_TRACE_COLL_DS	Trace Collector Download Station.
DEV_TSL	internal use only! don't use!

**BPNGErrCode**

enum [BPNGErrCode](#)

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)

## Enumerator

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 <code>IBPNGClient::initOnline</code> for data download or conversion directly from the device and <code>IBPNGClient::initOffline</code> 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 <code>IBPNGClient::getNumConversionErrors()</code> and <code>IBPNGClient::getConversionError()</code> for further information
BPNG_CONV_SET_NOT_FOUND	The passed conversion set pointer was not created with this <code>IBPNGClient</code> instance and dus could not be found.
BPNG_NOTHING_TO_CONVERT	There is no data available that could be converted. Check the specified time/id spans.
BPNG_TMT_FILE_ID_ERROR	Invalid TMT/XTMT file id while trying to convert data.
BPNG_TMT_FORMAT_ERROR_VERSION	The TMT/XTMT version of the trace file is not supported by this lib version.
BPNG_TMT_FORMAT_ERROR_TS	Missing FileTimeMessage in header of TMT/XTMT file.
BPNG_INVALID_MESSAGE_ERROR	Invalid messages found in trace file(s).
BPNG_INVALID_MESSAGE_ID	Invalid message id found in trace file(s).
BPNG_INVALID_MESSAGE_TS	Invalid message ts found in trace file(s).
BPNG_INVALID_MESSAGE_SUBID	Invalid message sub id found in trace file(s).
BPNG_INVALID_MESSAGE_LEN	Invalid message length found in trace file(s).
BPNG_CONV_FORMAT_ERROR	Invalid format assignment or mismatching recorded trace data for the specified conversion format.
BPNG_DOWNLOAD_ERRORS	Multiple download errors. Use <code>IBPNGClient::getNumDownloadErrors()</code> and <code>IBPNGClient::getDownloadError()</code> 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 <code>IBPNGClient</code> from the passed offline data set.
BPNG_PARAMETER_MISMATCH	currently not used

**Enumerator**

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 reuired min version written to <a href="#">BPNGError::msg</a> .
BPNG_EXCEPTION	Some kind of unhandled exception was thrown.
BPNG_INCOMPATIBLE_RDB	The logger's or offline data set's RDB-Verison 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	No trace data found on download station. STATION
BPNG_EXCEPTION_MINI_DUMP	Some kind of unhandled exception was thrown. MiniDump file was written.

**BPNGLoggerStatus**

```
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 hast 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.

**BPNGWarningCode**

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 Protocol 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_TIME_ZONES	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.

**ChannelType**

enum [ChannelType](#)

Currently supported bus 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.

**Enumerator**

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.

**LanguageID**

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

**Reason**

enum [Reason](#)

List of possible reasons why a measure signal was ignored.

**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

[IBPNGClientListener](#) wrapper.

```
#include <vector>
#include <IBPNGClientListener.h>
#include <IBPNGClient.h>
#include <BPNGDefines.h>
#include <OnlineLoggerInfoWrapper.hh>
#include <TSLClusterImpl.hh>
```

## Classes

- class [BPNGLoggerDetector](#)

### 7.2.1 Detailed Description

[IBPNGClientListener](#) wrapper.

## 7.3 clientUtil.hh File Reference

Helper functions.

```
#include <chrono>
#include <iosfwd>
#include <iostream>
#include <sstream>
#include <time.h>
#include <string.h>
#include <vector>
#include <sys/stat.h>
```

## Functions

- const char \* [progrname](#) (const char \*path)  
*Get the program name, similar to basename.*
- template<class Duration >  
constexpr std::chrono::hours **toHours** (const Duration &d)
- template<class Duration >  
constexpr std::chrono::minutes **toMinutes** (const Duration &d)
- template<class Duration >  
constexpr std::chrono::seconds **toSeconds** (const Duration &d)
- template<class Duration >  
constexpr std::chrono::milliseconds **toMilliseconds** (const Duration &d)
- template<class Duration >  
constexpr std::chrono::microseconds **toMicroseconds** (const Duration &d)
- template<class Duration >  
constexpr std::chrono::nanoseconds **toNanoseconds** (const Duration &d)
- template<typename Rep, typename Period >  
timespec [toTimespec](#) (const std::chrono::duration< Rep, Period > &d)  
*Convert a chrono duration into a POSIX timespec.*
- static void **printTm** (std::ostream &os, time\_t sec, char origfill)
- std::ostream & **operator**<< (std::ostream &os, const tm &\_t)
- std::ostream & [operator](#)<< (std::ostream &os, const timespec &\_t)

*Output operator for struct timespec as used e.g. by pselect or nanosleep.*

- `template<typename Rep , typename Period >`  
`std::ostream & operator<< (std::ostream &os, const std::chrono::duration< Rep, Period >`  
`&d)`

*Output operator for a chrono duration.*

- `template<typename Rep >`  
`std::ostream & operator<< (std::ostream &os, const std::chrono::duration< Rep, std::ratio<`  
`1 >> &d)`

*Output operator for a chrono duration with period one.*

- `template<typename Rep >`  
`std::ostream & operator<< (std::ostream &os, const std::chrono::duration< Rep, std::milli`  
`> &d)`

*Output operator for a chrono duration with Period milli.*

- `template<typename Rep >`  
`std::ostream & operator<< (std::ostream &os, const std::chrono::duration< Rep, std::micro`  
`> &d)`

*Output operator for a chrono duration with Period micro.*

- `template<typename Rep >`  
`std::ostream & operator<< (std::ostream &os, const std::chrono::duration< Rep, std::nano`  
`> &d)`

*Output operator for a chrono duration with Period nano.*

- `template<typename Clock , typename Duration >`  
`std::ostream & operator<< (std::ostream &os, const std::chrono::time_point< Clock, Dura-`  
`tion > &tp)`
- `int mkdirPath (const std::string &_dirpath, int _mode=S_IRWXU|S_IRWXG|S_IRWXO)`

### 7.3.1 Detailed Description

Helper functions.

### 7.3.2 Function Documentation

#### **operator<<()** [1/7]

```
std::ostream& operator<< (
    std::ostream & os,
    const timespec & _t )
```

Output operator for struct timespec as used e.g. by pselect or nanosleep.  
 Prints out the time specified in `_t` in seconds.

#### **operator<<()** [2/7]

```
template<typename Rep , typename Period >
std::ostream& operator<< (
    std::ostream & os,
    const std::chrono::duration< Rep, Period > & d ) [inline]
```

Output operator for a chrono duration.  
 If the duration has to be printed in a hh:mm:ss.ms notation you have to use e.g.  
`cout << toTimespec(d)`



**operator<<()** [3/7]

```
template<typename Rep >
std::ostream& operator<< (
    std::ostream & os,
    const std::chrono::duration< Rep, std::ratio< 1 >> & d )
```

Output operator for a chrono duration with period one.

Formatting:

- `std::ios::showbase`: the unit "s" will be printed
- `std::ios::skipws`: no space between the time and the unit

**operator<<()** [4/7]

```
template<typename Rep >
std::ostream& operator<< (
    std::ostream & os,
    const std::chrono::duration< Rep, std::milli > & d )
```

Output operator for a chrono duration with Period milli.

Formatting:

- `std::ios::showbase`: the unit "ms" will be printed
- `std::ios::skipws`: no space between the time and the unit

**operator<<()** [5/7]

```
template<typename Rep >
std::ostream& operator<< (
    std::ostream & os,
    const std::chrono::duration< Rep, std::micro > & d )
```

Output operator for a chrono duration with Period micro.

Formatting:

- `std::ios::showbase`: the unit "µs" will be printed
- `std::ios::skipws`: no space between the time and the unit

**operator<<()** [6/7]

```
template<typename Rep >
std::ostream& operator<< (
    std::ostream & os,
    const std::chrono::duration< Rep, std::nano > & d )
```

Output operator for a chrono duration with Period nano.

Formatting:

- `std::ios::showbase`: the unit "ns" will be printed
- `std::ios::skipws`: no space between the time and the unit

**operator<<()** [7/7]

```
template<typename Clock , typename Duration >
std::ostream& operator<< (
    std::ostream & os,
    const std::chrono::time_point< Clock, Duration > & tp ) [inline]
```

Output operator for a chrono time point. Prints the date and time since the epoch. The following adjustments can be done:

- If `std::ios_base::scientific` is set in the flags the duration will be printed seconds plus fractions of a second, otherwise it will be printed in a readable format.
- If the precision is set to a value of 9 or larger nanosecond precision is printed.

## 7.4 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](#) ()

### 7.4.1 Detailed Description

Interface class for the BPNGClient DLL.

Author

Markus van Pinxteren

Date

21.04.2010

### 7.4.2 Function Documentation

#### **addLogListener()**

```
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](#)

#### **createEmptyOnlineLoggerInfo()**

```
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](#)

**createNewClientProperties()**

```
DECLDIR IClientProperties* WINAPI createNewClientProperties ( )
```

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

See also

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

**freeClientPropertiesMemory()**

```
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.

**freeConversionSetMemory()**

```
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.

**getBPNGClient()**

```
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\(\)](#)

**getNumTSLMemberFromOfflineDataSet()**

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

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

**writeLogFile()**

```
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.

### **writeLogToCout()**

```
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.

## **7.5 IBPNGClientListener.h File Reference**

Interface class for the BPNGClient listener.

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

### **Classes**

- struct [IBPNGClientListener](#)

#### **7.5.1 Detailed Description**

Interface class for the BPNGClient listener.

Author

Markus van Pinxteren

Date

12.05.2010

## **7.6 IClientProperties.h File Reference**

Interface for client properties.

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

### **Classes**

- struct [IClientProperties](#)

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

#### **7.6.1 Detailed Description**

Interface for client properties.

Author

Markus van Pinxteren

Date

20.03.2014

## 7.7 OnlineLoggerInfoWrapper.hh File Reference

C++ wrapper around brain dead [OnlineLoggerInfo](#).

```
#include <cstdlib>
#include <cstring>
#include <vector>
#include <iosfwd>
#include <IBPNGClient.h>
```

### Classes

- class [OnlineLoggerInfoWrapper](#)  
*Wrapper around brain dead [OnlineLoggerInfo](#).*

### Functions

- `std::ostream & operator<< (std::ostream &os, const OnlineLoggerInfoWrapper &oli)`
- `std::ostream & operator<< (std::ostream &os, const std::vector< OnlineLoggerInfoWrapper > &list)`

#### 7.7.1 Detailed Description

C++ wrapper around brain dead [OnlineLoggerInfo](#).

## 7.8 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 }

### 7.8.1 Detailed Description

Public interfaces for Telemotive Reference Database access.

### 7.8.2 Enumeration Type Documentation

#### RdbEventType

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

**Enumerator**

TESTDRIVE_INFO	misc. info event for test drive data like testname, vin, map-version, reproducibility, etc.
----------------	---

## 7.9 RdbEventList.hh File Reference

[IRdbEvent](#) wrapper.

```
#include <vector>
#include <string>
#include "BPNGDefines.h"
#include "RdbDefines.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.*

### 7.9.1 Detailed Description

[IRdbEvent](#) wrapper.

## 7.10 RdbTraceBlockList.hh File Reference

[IRdbTraceBlock](#) wrapper.

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

### Classes

- struct [RdbTraceBlock2](#)
- class [RdbTraceBlockList](#)

### 7.10.1 Detailed Description

[IRdbTraceBlock](#) wrapper.



## 7.11 TSLClusterImpl.hh File Reference

TSLCluster wrapper.

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

### Classes

- class [TSLClusterImpl](#)

#### 7.11.1 Detailed Description

TSLCluster wrapper.



# Index

activateGatewayLoggerDetection  
     IBPNGClient, [41](#)  
 addAnalogPortSettings  
     IClientProperties, [71](#)  
 addChannel  
     IConversionSet, [76](#)  
 addDevice  
     TSLClusterImpl, [92](#)  
 addDigitalPortSettings  
     IClientProperties, [71](#)  
 addLogListener  
     IBPNGClient.h, [111](#)  
 addRdbldRange  
     IConversionSet, [76](#)  
 addTimeSpan  
     IConversionSet, [77](#)  
 assignDBCFile  
     IBPNGClient, [42](#)  
 assignDatabaseFile  
     IBPNGClient, [42](#)  
  
 BPNGBugreportMode  
     BPNGDefines.h, [100](#)  
 BPNGDefines.h, [95](#)  
     BPNGBugreportMode, [100](#)  
     BPNGDeviceType, [101](#)  
     BPNGErrCode, [101](#)  
     BPNGLoggerStatus, [104](#)  
     BPNGWarningCode, [104](#)  
     ChannelType, [105](#)  
     LanguageID, [106](#)  
     Reason, [106](#)  
 BPNGDeviceType  
     BPNGDefines.h, [101](#)  
 BPNGErrCode  
     BPNGDefines.h, [101](#)  
 BPNGError, [29](#)  
 BPNGLoggerDetector, [29](#)  
     excludeRCTFromTSL, [31](#)  
     getLoggerList, [31](#)  
     getOverwritingPermission, [31](#)  
     getTSLs, [31](#)  
     onBPNGDeviceDetected, [32](#)  
     onBPNGDeviceDisappeared, [32](#)  
     onBPNGDeviceStateChange, [32](#)

    onConversionStart, [32](#)  
     onCriticalDiskSpace, [33](#)  
     onDataRecoverProgress, [33](#)  
     onDownloadStart, [33](#)  
     onExtractionPasswordRequired, [34](#)  
     onGetLogReportProgress, [34](#)  
     onInvalidPwConfigFound, [34](#)  
     onProgressConversion, [34](#)  
     onProgressDataDownload, [35](#)  
     onProgressDeletion, [36](#)  
     onStatusMessage, [36](#)  
     onTargetPathTooLong, [36](#)  
     onWarning, [37](#)  
 BPNGLoggerDetector.hh, [106](#)  
 BPNGLoggerStatus  
     BPNGDefines.h, [104](#)  
 BPNGWarningCode  
     BPNGDefines.h, [104](#)  
 begin  
     TSLClusterImpl, [93](#)  
 BPNG\_AMBIGUOUS\_IP  
     BPNGDefines.h, [102](#)  
 BPNG\_CONNECT\_FTP\_FAILED  
     BPNGDefines.h, [102](#)  
 BPNG\_CONNECT\_TMPBUS\_FAILED  
     BPNGDefines.h, [102](#)  
 BPNG\_CONV\_FORMAT\_ERROR  
     BPNGDefines.h, [103](#)  
 BPNG\_CONV\_SET\_NOT\_FOUND  
     BPNGDefines.h, [103](#)  
 BPNG\_CONVERSION\_ERRORS  
     BPNGDefines.h, [103](#)  
 BPNG\_DIR\_EXISTS\_ERROR  
     BPNGDefines.h, [102](#)  
 BPNG\_DOWNLOAD\_ERRORS  
     BPNGDefines.h, [103](#)  
 BPNG\_ENGLISH  
     BPNGDefines.h, [106](#)  
 BPNG\_EXCEPTION  
     BPNGDefines.h, [104](#)  
 BPNG\_EXCEPTION\_MINI\_DUMP  
     BPNGDefines.h, [104](#)  
 BPNG\_FAILED\_TO\_CONNECT\_STREAMING  
     BPNGDefines.h, [102](#)  
 BPNG\_FAILED\_TO\_CREATE\_LOCAL\_FILE\_OR\_DIRECTORY

- BPNGDefines.h, [102](#)
- BPNG\_FILE\_EXISTS\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_CHANGE\_CWD\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_CREATE\_REMOTE\_DIR\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_LOGIN\_FAILED
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_NOT\_CONNECTED
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_READ\_REMOTE\_FILE\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_REMOTE\_PATH\_NOT\_FOUND
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_REMOVE\_REMOTE\_DIR\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_REMOVE\_REMOTE\_FILE\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_SERVER\_NOT\_FOUND
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_TRANSFER\_USER\_CANCELED
  - BPNGDefines.h, [102](#)
- BPNG\_FTP\_WRITE\_REMOTE\_FILE\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_FW\_VERSION\_CHECK\_ERROR
  - BPNGDefines.h, [104](#)
- BPNG\_GERMAN
  - BPNGDefines.h, [106](#)
- BPNG\_INCOMPATIBLE\_RDB
  - BPNGDefines.h, [104](#)
- BPNG\_INCONSISTENT\_TSL\_FWVERSIONS
  - BPNGDefines.h, [104](#)
- BPNG\_INITIALISATION\_ERROR
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_MESSAGE\_ERROR
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_MESSAGE\_ID
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_MESSAGE\_LEN
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_MESSAGE\_SUBID
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_MESSAGE\_TS
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_OFFLINE\_SET
  - BPNGDefines.h, [103](#)
- BPNG\_INVALID\_RPC\_COMMAND
  - BPNGDefines.h, [104](#)
- BPNG\_INVALID\_TSL\_CASCDING
  - BPNGDefines.h, [104](#)
- BPNG\_INVALID\_TSL\_CLUSTER
  - BPNGDefines.h, [104](#)
- BPNG\_LOCAL\_PATH\_NOT\_FOUND
  - BPNGDefines.h, [102](#)
- BPNG\_LOGGER\_NOT\_FOUND
  - BPNGDefines.h, [102](#)
- BPNG\_MIN\_VERSION\_ERROR
  - BPNGDefines.h, [104](#)
- BPNG\_NO\_TRACE\_DATA\_ON\_DOWNLOAD\_STATION
  - BPNGDefines.h, [104](#)
- BPNG\_NOERR
  - BPNGDefines.h, [102](#)
- BPNG\_NOT\_CONNECTED
  - BPNGDefines.h, [102](#)
- BPNG\_NOTHING\_TO\_CONVERT
  - BPNGDefines.h, [103](#)
- BPNG\_NOTHING\_TO\_DOWNLOAD
  - BPNGDefines.h, [103](#)
- BPNG\_NOTHING\_TO\_TEST\_REPORT
  - BPNGDefines.h, [104](#)
- BPNG\_PARAMETER\_MISMATCH
  - BPNGDefines.h, [103](#)
- BPNG\_RDB\_OPEN\_FAILED
  - BPNGDefines.h, [103](#)
- BPNG\_RDB\_SQLITE\_QUERY\_ERROR
  - BPNGDefines.h, [103](#)
- BPNG\_READ\_LOCAL\_FILE\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_TARGET\_PATH\_TOO\_LONG
  - BPNGDefines.h, [102](#)
- BPNG\_TMPBUS\_COPYRDB\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_TMPBUS\_NOT\_CONNECTED
  - BPNGDefines.h, [102](#)
- BPNG\_TMPBUS\_REQUEST\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_TMPBUS\_SEND\_MSG\_ERROR
  - BPNGDefines.h, [102](#)
- BPNG\_TMT\_FILE\_ID\_ERROR
  - BPNGDefines.h, [103](#)
- BPNG\_TMT\_FORMAT\_ERROR\_TS
  - BPNGDefines.h, [103](#)
- BPNG\_TMT\_FORMAT\_ERROR\_VERSION
  - BPNGDefines.h, [103](#)
- BPNG\_UNSPECIFIED\_ERROR
  - BPNGDefines.h, [104](#)
- BPNG\_USER\_CANCELLED
  - BPNGDefines.h, [104](#)
- BPNG\_WARNING\_CLOSE\_TRACE\_FILES
  - BPNGDefines.h, [105](#)
- BPNG\_WARNING\_MESSAGES\_NOT\_CONVERTED
  - BPNGDefines.h, [105](#)
- BPNG\_WARNING\_NO\_ESO\_TRACE
  - BPNGDefines.h, [105](#)
- BPNG\_WARNING\_RECOVERING\_FAILED
  - BPNGDefines.h, [105](#)
- BPNG\_WARNING\_TSL\_WITH\_DIFFERENT\_TIMEZONES

- BPNGDefines.h, 105
- BPNG\_WRITE\_LOCAL\_FILE\_ERROR
  - BPNGDefines.h, 102
- BPNG\_XML\_PARSER\_ERROR
  - BPNGDefines.h, 103
- BPNG\_ZIP\_EXCEEDS\_FATFS\_MAX
  - BPNGDefines.h, 103
- BPNGDefines.h
  - BPNG\_AMBIGUOUS\_IP, 102
  - BPNG\_CONNECT\_FTP\_FAILED, 102
  - BPNG\_CONNECT\_TMPBUS\_FAILED, 102
  - BPNG\_CONV\_FORMAT\_ERROR, 103
  - BPNG\_CONV\_SET\_NOT\_FOUND, 103
  - BPNG\_CONVERSION\_ERRORS, 103
  - BPNG\_DIR\_EXISTS\_ERROR, 102
  - BPNG\_DOWNLOAD\_ERRORS, 103
  - BPNG\_ENGLISH, 106
  - BPNG\_EXCEPTION, 104
  - BPNG\_EXCEPTION\_MINI\_DUMP, 104
  - BPNG\_FAILED\_TO\_CONNECT\_STREAMING, 102
  - BPNG\_FAILED\_TO\_CREATE\_LOCAL\_FILE\_OR\_DIRECTORY, 102
  - BPNG\_FILE\_EXISTS\_ERROR, 102
  - BPNG\_FTP\_CHANGE\_CWD\_ERROR, 102
  - BPNG\_FTP\_CREATE\_REMOTE\_DIR\_ERROR, 102
  - BPNG\_FTP\_LOGIN\_FAILED, 102
  - BPNG\_FTP\_NOT\_CONNECTED, 102
  - BPNG\_FTP\_READ\_REMOTE\_FILE\_ERROR, 102
  - BPNG\_FTP\_REMOTE\_PATH\_NOT\_FOUND, 102
  - BPNG\_FTP\_REMOVE\_REMOTE\_DIR\_ERROR, 102
  - BPNG\_FTP\_REMOVE\_REMOTE\_FILE\_ERROR, 102
  - BPNG\_FTP\_SERVER\_NOT\_FOUND, 102
  - BPNG\_FTP\_TRANSFER\_USER\_CANCELED, 102
  - BPNG\_FTP\_WRITE\_REMOTE\_FILE\_ERROR, 102
  - BPNG\_FW\_VERSION\_CHECK\_ERROR, 104
  - BPNG\_GERMAN, 106
  - BPNG\_INCOMPATIBLE\_RDB, 104
  - BPNG\_INCONSISTENT\_TSL\_FWVERSIONS, 104
  - BPNG\_INITIALISATION\_ERROR, 103
  - BPNG\_INVALID\_MESSAGE\_ERROR, 103
  - BPNG\_INVALID\_MESSAGE\_ID, 103
  - BPNG\_INVALID\_MESSAGE\_LEN, 103
  - BPNG\_INVALID\_MESSAGE\_SUBID, 103
  - BPNG\_INVALID\_MESSAGE\_TS, 103
  - BPNG\_INVALID\_OFFLINE\_SET, 103
  - BPNG\_INVALID\_RPC\_COMMAND, 104
  - BPNG\_INVALID\_TSL\_CASCDING, 104
  - BPNG\_INVALID\_TSL\_CLUSTER, 104
  - BPNG\_LOCAL\_PATH\_NOT\_FOUND, 102
  - BPNG\_LOGGER\_NOT\_FOUND, 102
  - BPNG\_MIN\_VERSION\_ERROR, 104
  - BPNG\_NO\_TRACE\_DATA\_ON\_DOWNLOAD\_STATION, 104
  - BPNG\_NOERR, 102
  - BPNG\_NOT\_CONNECTED, 102
  - BPNG\_NOTHING\_TO\_CONVERT, 103
  - BPNG\_NOTHING\_TO\_DOWNLOAD, 103
  - BPNG\_NOTHING\_TO\_TEST\_REPORT, 104
  - BPNG\_PARAMETER\_MISMATCH, 103
  - BPNG\_RDB\_OPEN\_FAILED, 103
  - BPNG\_RDB\_SQLITE\_QUERY\_ERROR, 103
  - BPNG\_READ\_LOCAL\_FILE\_ERROR, 102
  - BPNG\_TARGET\_PATH\_TOO\_LONG, 102
  - BPNG\_TMPBUS\_COPYRDB\_ERROR, 102
  - BPNG\_TMPBUS\_NOT\_CONNECTED, 102
  - BPNG\_TMPBUS\_REQUEST\_ERROR, 102
  - BPNG\_TMPBUS\_SEND\_MSG\_ERROR, 102
  - BPNG\_TMT\_FILE\_ID\_ERROR, 103
  - BPNG\_TMT\_FORMAT\_ERROR\_TS, 103
  - BPNG\_TMT\_FORMAT\_ERROR\_VERSION, 103
  - BPNG\_UNSPECIFIED\_ERROR, 104
  - BPNG\_USER\_CANCELLED, 104
  - BPNG\_WARNING\_CLOSE\_TRACE\_FILES, 105
  - BPNG\_WARNING\_MESSAGES\_NOT\_CONVERTED, 105
  - BPNG\_WARNING\_NO\_ESO\_TRACE, 105
  - BPNG\_WARNING\_RECOVERING\_FAILED, 105
  - BPNG\_WARNING\_TSL\_WITH\_DIFFERENT\_TIMEZONES, 105
  - BPNG\_WRITE\_LOCAL\_FILE\_ERROR, 102
  - BPNG\_XML\_PARSER\_ERROR, 103
  - BPNG\_ZIP\_EXCEEDS\_FATFS\_MAX, 103
  - BR\_FDB\_RDB, 101
  - BR\_FULL\_ALL\_TRACES, 101
  - BR\_FULL\_TIMESPAN\_TRACES, 101
  - BR\_FULL\_WO\_TRACES, 101
  - BR\_ONLY\_CLIENT, 101
  - BR\_ONLY\_LOGS, 101
  - CH\_ANALOG\_IN, 105
  - CH\_CAMERA, 106
  - CH\_CAN, 105
  - CH\_CCPXCP, 106
  - CH\_DIAG, 106
  - CH\_DIGITAL\_IN, 106
  - CH\_ECL, 106
  - CH\_ETHERNET, 105

- CH\_FLEXRAY, [105](#)
- CH\_GPS, [106](#)
- CH\_LIN, [105](#)
- CH\_MII, [106](#)
- CH\_MOST150\_CTRL, [105](#)
- CH\_MOST150\_MDP, [105](#)
- CH\_MOST150\_MEP, [105](#)
- CH\_MOST150\_STREAM, [105](#)
- CH\_MOST25\_CTRL, [105](#)
- CH\_MOST25\_MDP, [105](#)
- CH\_MOST25\_SYNC, [105](#)
- CH\_SERIAL, [105](#)
- CH\_TTY, [106](#)
- CH\_UNDEFINED, [105](#)
- DEV\_BP2, [101](#)
- DEV\_BP2\_V1X, [101](#)
- DEV\_BP2\_V2X, [101](#)
- DEV\_BP\_REMOTE, [101](#)
- DEV\_BP\_TOUCH, [101](#)
- DEV\_BPMINI, [101](#)
- DEV\_RAPID, [101](#)
- DEV\_RC\_TOUCH, [101](#)
- DEV\_TRACE\_COLL, [101](#)
- DEV\_TRACE\_COLL\_DS, [101](#)
- DEV\_TSL, [101](#)
- LS\_ERROR, [104](#)
- LS\_FWUPDATE, [104](#)
- LS\_MEM, [104](#)
- LS\_NOSYNC, [104](#)
- LS\_OK, [104](#)
- LS\_RING, [104](#)
- LS\_WARNING, [104](#)
- OBSOLETE\_CH\_CANLS, [105](#)
- R\_BIT\_MASK\_OVERLAP, [106](#)
- R\_FORBIDDEN\_TAB\_VALUE, [106](#)
- R\_UNKNOWN, [106](#)
- R\_UNSUPPORTED\_BIT\_MASK, [106](#)
- R\_UNSUPPORTED\_COMPU\_TAB, [106](#)
- BR\_FDB\_RDB
  - BPNGDefines.h, [101](#)
- BR\_FULL\_ALL\_TRACES
  - BPNGDefines.h, [101](#)
- BR\_FULL\_TIMESPAN\_TRACES
  - BPNGDefines.h, [101](#)
- BR\_FULL\_WO\_TRACES
  - BPNGDefines.h, [101](#)
- BR\_ONLY\_CLIENT
  - BPNGDefines.h, [101](#)
- BR\_ONLY\_LOGS
  - BPNGDefines.h, [101](#)
- BUGREPORT
  - TSLClusterImpl, [92](#)
- CH\_ANALOG\_IN
  - BPNGDefines.h, [105](#)
- CH\_CAMERA
  - BPNGDefines.h, [106](#)
- CH\_CAN
  - BPNGDefines.h, [105](#)
- CH\_CCPXCP
  - BPNGDefines.h, [106](#)
- CH\_DIAG
  - BPNGDefines.h, [106](#)
- CH\_DIGITAL\_IN
  - BPNGDefines.h, [106](#)
- CH\_ECL
  - BPNGDefines.h, [106](#)
- CH\_ETHERNET
  - BPNGDefines.h, [105](#)
- CH\_FLEXRAY
  - BPNGDefines.h, [105](#)
- CH\_GPS
  - BPNGDefines.h, [106](#)
- CH\_LIN
  - BPNGDefines.h, [105](#)
- CH\_MII
  - BPNGDefines.h, [106](#)
- CH\_MOST150\_CTRL
  - BPNGDefines.h, [105](#)
- CH\_MOST150\_MDP
  - BPNGDefines.h, [105](#)
- CH\_MOST150\_MEP
  - BPNGDefines.h, [105](#)
- CH\_MOST150\_STREAM
  - BPNGDefines.h, [105](#)
- CH\_MOST25\_CTRL
  - BPNGDefines.h, [105](#)
- CH\_MOST25\_MDP
  - BPNGDefines.h, [105](#)
- CH\_MOST25\_SYNC
  - BPNGDefines.h, [105](#)
- CH\_SERIAL
  - BPNGDefines.h, [105](#)
- CH\_TTY
  - BPNGDefines.h, [106](#)
- CH\_UNDEFINED
  - BPNGDefines.h, [105](#)
- ChannelType
  - BPNGDefines.h, [105](#)
- clearDBCFileAssignments
  - IBPNGClient, [42](#)
- clientUtil.hh, [107](#)
  - operator<<, [108](#), [109](#)
- CONFIG
  - RdbDefines.h, [115](#)
  - TSLClusterImpl, [92](#)
- connectLogger
  - IBPNGClient, [42](#)

- ConnectionType
  - TSLClusterImpl, 92
- CONVERSION
  - TSLClusterImpl, 92
- convertData
  - IBPNGClient, 43
- createEmptyOnlineLoggerInfo
  - IBPNGClient.h, 111
- createNewClientProperties
  - IBPNGClient.h, 111
- createNewConversionSet
  - IBPNGClient, 43
- createTestReport
  - IBPNGClient, 43
- DataSpan, 37
- DATA\_DELETED
  - RdbDefines.h, 115
- deleteAllData
  - IBPNGClient, 44
- deleteDevice
  - TSLClusterImpl, 93
- deleteSectionsByStartUplds
  - IBPNGClient, 44
- DEV\_BP2
  - BPNGDefines.h, 101
- DEV\_BP2\_V1X
  - BPNGDefines.h, 101
- DEV\_BP2\_V2X
  - BPNGDefines.h, 101
- DEV\_BP\_REMOTE
  - BPNGDefines.h, 101
- DEV\_BP\_TOUCH
  - BPNGDefines.h, 101
- DEV\_BPMINI
  - BPNGDefines.h, 101
- DEV\_RAPID
  - BPNGDefines.h, 101
- DEV\_RC\_TOUCH
  - BPNGDefines.h, 101
- DEV\_TRACE\_COLL
  - BPNGDefines.h, 101
- DEV\_TRACE\_COLL\_DS
  - BPNGDefines.h, 101
- DEV\_TSL
  - BPNGDefines.h, 101
- deviceType
  - OnlineLoggerInfo, 85
- DOWNLOAD
  - TSLClusterImpl, 92
- downloadBugReport
  - IBPNGClient, 45
- downloadDataSpans
  - IBPNGClient, 45
- enableClientLogOutput
  - IBPNGClient, 46
- end
  - TSLClusterImpl, 93
- excludeRCTFromTSL
  - BPNGLoggerDetector, 31
- filterSignals
  - IBPNGClient, 46
- filterSignalsFromOfflineData
  - IBPNGClient, 46
- flashDeviceLED
  - IBPNGClient, 47
- freeClientPropertiesMemory
  - IBPNGClient.h, 112
- freeConversionSetMemory
  - IBPNGClient.h, 112
- FW\_UPDATE
  - TSLClusterImpl, 92
- getAvailableFormats
  - IBPNGClient, 47
- getBPNGClient
  - IBPNGClient.h, 112
- getClientProperties
  - IBPNGClient, 47
- getComment
  - IRdbEvent, 80
- getConfig
  - IBPNGClient, 47
- getConfigPath
  - IBPNGClient, 48
- getConversionError
  - IBPNGClient, 48
- getDeviceName
  - IBPNGClient, 48
- getDownloadError
  - IBPNGClient, 48
- getEventList
  - IBPNGClient, 49
- getFalseMeasureSignals
  - IBPNGClient, 49
- getLastError
  - IBPNGClient, 49
- getLicenses
  - IBPNGClient, 49
- getLoggerChannels
  - IBPNGClient, 50
- getLoggerList
  - BPNGLoggerDetector, 31
- getMemoryFillLevel
  - IBPNGClient, 50
- getNumConversionErrors
  - IBPNGClient, 50

- getNumDownloadErrors
  - IBPNGClient, 50
- getNumTSLMemberFromOfflineDataSet
  - IBPNGClient.h, 112
- getOverwritingPermission
  - BPNGLoggerDetector, 31
  - IBPNGClientListener, 60
- getPwdFile
  - IBPNGClient, 51
- getReferenceDataBasePath
  - IBPNGClient, 51
- getTSLName
  - TSLClusterImpl, 93
- getTSLs
  - BPNGLoggerDetector, 31
- getTimeZone
  - IRdbEvent, 80
- getTraceBlockList
  - IBPNGClient, 51
- getUniqueld
  - IRdbEvent, 80
- getVersions
  - IBPNGClient, 52
- IBPNGClient, 37
  - activateGatewayLoggerDetection, 41
  - assignDBCFile, 42
  - assignDatabaseFile, 42
  - clearDBCFileAssignments, 42
  - connectLogger, 42
  - convertData, 43
  - createNewConversionSet, 43
  - createTestReport, 43
  - deleteAllData, 44
  - deleteSectionsByStartUpIds, 44
  - downloadBugReport, 45
  - downloadDataSpans, 45
  - enableClientLogOutput, 46
  - filterSignals, 46
  - filterSignalsFromOfflineData, 46
  - flashDeviceLED, 47
  - getAvailableFormats, 47
  - getClientProperties, 47
  - getConfig, 47
  - getConfigPath, 48
  - getConversionError, 48
  - getDeviceName, 48
  - getDownloadError, 48
  - getEventList, 49
  - getFalseMeasureSignals, 49
  - getLastError, 49
  - getLicenses, 49
  - getLoggerChannels, 50
  - getMemoryFillLevel, 50
  - getNumConversionErrors, 50
  - getNumDownloadErrors, 50
  - getPwdFile, 51
  - getReferenceDataBasePath, 51
  - getTraceBlockList, 51
  - getVersions, 52
  - initialize, 52
  - isPasswordProtectionSupported, 52
  - keepLoggerAlive, 53
  - reconfigLogger, 53
  - release, 54
  - removeAllLicenses, 54
  - restartDevice, 54
  - scanNetworkForLogger, 55
  - setClientProperties, 55
  - setDefaultConfig, 55
  - setDevice, 55
  - setInfoEvent, 56
  - setMarker, 56
  - setOfflineData, 56
  - setPwdFile, 56
  - setTSLCluster, 57
  - setTime, 57
  - shutdownDevice, 57
  - startLiveDownload, 57
  - synchronizeRdb, 58
  - updateFirmware, 58
  - updateLicenses, 58
- IBPNGClient.h, 110
  - addLogListener, 111
  - createEmptyOnlineLoggerInfo, 111
  - createNewClientProperties, 111
  - freeClientPropertiesMemory, 112
  - freeConversionSetMemory, 112
  - getBPNGClient, 112
  - getNumTSLMemberFromOfflineDataSet, 112
  - writeLogFile, 112
  - writeLogToCout, 113
- IBPNGClientListener, 59
  - getOverwritingPermission, 60
  - onBPNGDeviceDetected, 60
  - onBPNGDeviceDisappeared, 60
  - onBPNGDeviceStateChange, 61
  - onConversionStart, 61
  - onCriticalDiskSpace, 61
  - onDataRecoverProgress, 62
  - onDownloadStart, 62
  - onExtractionPasswordRequired, 62
  - onGetLogReportProgress, 62
  - onInvalidPwConfigFound, 63
  - onLogInDataRequired, 63
  - onProgressConversion, 63
  - onProgressDataDownload, 64
  - onProgressDeletion, 65



- onStatusMessage, 65
- onTargetPathTooLong, 65
- onWarning, 65
- IBPNGClientListener.h, 113
- IChannel, 66
- IChannelList, 66
- IClientProperties, 67
  - addAnalogPortSettings, 71
  - addDigitalPortSettings, 71
  - setCANPseudoMsgTimeStampProperties, 72
  - setCANPseudoMsgTriggerProperties, 72
  - setCommonProperties, 73
  - setFlexRayPseudoMsgProperties, 74
  - setMOSTPseudoMsgProperties, 74
- IClientProperties.h, 113
- IConversionSet, 75
  - addChannel, 76
  - addRdbldRange, 76
  - addTimeSpan, 77
  - loadConversionFilters, 77
- IFalseMeasureSignal, 77
- IFalseMeasureSignalList, 78
- IFormatInfo, 78
- IFormatList, 79
- IRdbEvent, 79
  - getComment, 80
  - getTimeZone, 80
  - getUniqueId, 80
- IRdbEventList, 80
- IRdbTraceBlock, 81
- IRdbTraceBlockList, 81
- ITesttoolsChannel, 81
- ITesttoolsChannelList, 82
- INFO
  - RdbDefines.h, 115
- initialize
  - IBPNGClient, 52
- isPasswordProtectionSupported
  - IBPNGClient, 52
- keepLoggerAlive
  - IBPNGClient, 53
- LanguageID
  - BPNGDefines.h, 106
- loadConversionFilters
  - IConversionSet, 77
- LogInData, 83
- loggerStatus
  - OnlineLoggerInfo, 85
- LS\_ERROR
  - BPNGDefines.h, 104
- LS\_FWUPDATE
  - BPNGDefines.h, 104
- LS\_MEM
  - BPNGDefines.h, 104
- LS\_NOSYNC
  - BPNGDefines.h, 104
- LS\_OK
  - BPNGDefines.h, 104
- LS\_RING
  - BPNGDefines.h, 104
- LS\_WARNING
  - BPNGDefines.h, 104
- MARKER
  - RdbDefines.h, 115
- MemoryFillLevel, 83
- NEW\_TIME
  - RdbDefines.h, 115
- OBSOLETE\_CH\_CANLS
  - BPNGDefines.h, 105
- onBPNGDeviceDetected
  - BPNGLoggerDetector, 32
  - IBPNGClientListener, 60
- onBPNGDeviceDisappeared
  - BPNGLoggerDetector, 32
  - IBPNGClientListener, 60
- onBPNGDeviceStateChange
  - BPNGLoggerDetector, 32
  - IBPNGClientListener, 61
- onConversionStart
  - BPNGLoggerDetector, 32
  - IBPNGClientListener, 61
- onCriticalDiskSpace
  - BPNGLoggerDetector, 33
  - IBPNGClientListener, 61
- onDataRecoverProgress
  - BPNGLoggerDetector, 33
  - IBPNGClientListener, 62
- onDownloadStart
  - BPNGLoggerDetector, 33
  - IBPNGClientListener, 62
- onExtractionPasswordRequired
  - BPNGLoggerDetector, 34
  - IBPNGClientListener, 62
- onGetLogReportProgress
  - BPNGLoggerDetector, 34
  - IBPNGClientListener, 62
- onInvalidPwConfigFound
  - BPNGLoggerDetector, 34
  - IBPNGClientListener, 63
- onLogInDataRequired
  - IBPNGClientListener, 63
- onProgressConversion
  - BPNGLoggerDetector, 34

- IBPNGClientListener, 63
- onProgressDataDownload
  - BPNGLoggerDetector, 35
  - IBPNGClientListener, 64
- onProgressDeletion
  - BPNGLoggerDetector, 36
  - IBPNGClientListener, 65
- onStatusMessage
  - BPNGLoggerDetector, 36
  - IBPNGClientListener, 65
- onTargetPathTooLong
  - BPNGLoggerDetector, 36
  - IBPNGClientListener, 65
- onWarning
  - BPNGLoggerDetector, 37
  - IBPNGClientListener, 65
- OnlineLoggerInfo, 84
  - deviceType, 85
  - loggerStatus, 85
- OnlineLoggerInfoStringPair, 86
- OnlineLoggerInfoWrapper, 86
- OnlineLoggerInfoWrapper.hh, 114
- operator<<
  - clientUtil.hh, 108, 109
- R\_BIT\_MASK\_OVERLAP
  - BPNGDefines.h, 106
- R\_FORBIDDEN\_TAB\_VALUE
  - BPNGDefines.h, 106
- R\_UNKNOWN
  - BPNGDefines.h, 106
- R\_UNSUPPORTED\_BIT\_MASK
  - BPNGDefines.h, 106
- R\_UNSUPPORTED\_COMPU\_TAB
  - BPNGDefines.h, 106
- RdbDefines.h, 114
  - RdbEventType, 115
- RdbEvent2, 88
- RdbEventList, 88
- RdbEventList.hh, 116
- RdbEventType
  - RdbDefines.h, 115
- RdbTraceBlock2, 89
- RdbTraceBlockList, 90
- RdbTraceBlockList.hh, 116
- RdbDefines.h
  - CONFIG, 115
  - DATA\_DELETED, 115
  - INFO, 115
  - MARKER, 115
  - NEW\_TIME, 115
  - SHUTDOWN, 115
  - SLAVE\_OFFSET, 115
  - SLAVE\_TO\_MASTER, 115
  - START\_TESTDRIVE, 115
  - STARTUP, 115
  - STOP\_TESTDRIVE, 115
  - SUDDEN\_DEATH, 115
  - TESTDRIVE\_INFO, 116
  - TIME\_SET, 115
  - TSL\_SESSION\_END, 115
  - TSL\_SESSION\_START, 115
  - TSL\_SLAVE\_OFFSET, 115
  - TSL\_SLAVE\_TO\_MASTER, 115
  - WAKEUP, 115
- Reason
  - BPNGDefines.h, 106
- reconfigLogger
  - IBPNGClient, 53
- release
  - IBPNGClient, 54
- removeAllLicenses
  - IBPNGClient, 54
- restartDevice
  - IBPNGClient, 54
- scanNetworkForLogger
  - IBPNGClient, 55
- setCANPseudoMsgTimeStampProperties
  - IClientProperties, 72
- setCANPseudoMsgTriggerProperties
  - IClientProperties, 72
- setClientProperties
  - IBPNGClient, 55
- setCommonProperties
  - IClientProperties, 73
- setDefaultConfig
  - IBPNGClient, 55
- setDevice
  - IBPNGClient, 55
- setFlexRayPseudoMsgProperties
  - IClientProperties, 74
- setInfoEvent
  - IBPNGClient, 56
- setMOSTPseudoMsgProperties
  - IClientProperties, 74
- setMarker
  - IBPNGClient, 56
- setOfflineData
  - IBPNGClient, 56
- setPwdFile
  - IBPNGClient, 56
- setTSLCluster
  - IBPNGClient, 57
- setTime
  - IBPNGClient, 57
- SHUTDOWN
  - RdbDefines.h, 115

- shutdownDevice
  - IBPNGClient, [57](#)
- SLAVE\_OFFSET
  - RdbDefines.h, [115](#)
- SLAVE\_TO\_MASTER
  - RdbDefines.h, [115](#)
- startLiveDownload
  - IBPNGClient, [57](#)
- START\_TESTDRIVE
  - RdbDefines.h, [115](#)
- STARTUP
  - RdbDefines.h, [115](#)
- STOP\_TESTDRIVE
  - RdbDefines.h, [115](#)
- SUDDEN\_DEATH
  - RdbDefines.h, [115](#)
- synchronizeRdb
  - IBPNGClient, [58](#)
- TSLCluster, [91](#)
- TSLClusterImpl, [91](#)
  - addDevice, [92](#)
  - begin, [93](#)
  - ConnectionType, [92](#)
  - deleteDevice, [93](#)
  - end, [93](#)
  - getTSLName, [93](#)
  - TSLClusterImpl, [92](#)
- TSLClusterImpl.hh, [117](#)
- TESTDRIVE\_INFO
  - RdbDefines.h, [116](#)
- TIME\_SET
  - RdbDefines.h, [115](#)
- TSL\_SESSION\_END
  - RdbDefines.h, [115](#)
- TSL\_SESSION\_START
  - RdbDefines.h, [115](#)
- TSL\_SLAVE\_OFFSET
  - RdbDefines.h, [115](#)
- TSL\_SLAVE\_TO\_MASTER
  - RdbDefines.h, [115](#)
- TSLClusterImpl
  - BUGREPORT, [92](#)
  - CONFIG, [92](#)
  - CONVERSION, [92](#)
  - DOWNLOAD, [92](#)
  - FW\_UPDATE, [92](#)
- updateFirmware
  - IBPNGClient, [58](#)
- updateLicenses
  - IBPNGClient, [58](#)
- WAKEUP
  - RdbDefines.h, [115](#)
- writeLogFile
  - IBPNGClient.h, [112](#)
- writeLogToCout
  - IBPNGClient.h, [113](#)