



Telemotive Client Library 3.3.1

User's manual

Generated by Doxygen 1.8.0

Mon Dec 11 2017 18:11:31

Inhaltsverzeichnis

1	User's manual - Telemotive Client Library 3.3.1	1
1.1	General	1
1.2	Functionality	1
1.3	Compiler/Linker	2
1.4	Thread safety	2
1.5	Demo project	2
2	Deprecated List	16
3	Hierarchical Index	17
3.1	Class Hierarchy	17
4	Class Index	18
4.1	Class List	18
5	File Index	20
5.1	File List	20
6	Class Documentation	21
6.1	BPNGError Struct Reference	21
6.2	BPNGLoggerDetector Class Reference	21
6.3	DataSpan Struct Reference	28
6.4	IBPNGClient Struct Reference	28
6.5	IBPNGClientListener Struct Reference	50
6.6	IChannel Struct Reference	55
6.7	IChannelList Struct Reference	56
6.8	IClientProperties Struct Reference	56
6.9	IConversionSet Struct Reference	62
6.10	IFalseMeasureSignal Struct Reference	64
6.11	IFalseMeasureSignalList Struct Reference	64
6.12	IFormatInfo Struct Reference	65
6.13	IFormatList Struct Reference	66
6.14	IRdbEvent Struct Reference	66
6.15	IRdbEventList Struct Reference	67
6.16	IRdbTraceBlock Struct Reference	67
6.17	IRdbTraceBlockList Struct Reference	68
6.18	ITesttoolsChannel Struct Reference	68
6.19	ITesttoolsChannelList Struct Reference	69
6.20	LogInData Struct Reference	70
6.21	MemoryFillLevel Struct Reference	70
6.22	OnlineLoggerInfo Struct Reference	71
6.23	OnlineLoggerInfoStringPair Struct Reference	72
6.24	RdbEvent2 Struct Reference	73

6.25 RdbEventList Class Reference	74
6.26 TSLCluster Class Reference	74
7 File Documentation	78
7.1 BPNGDefines.h File Reference	78
7.2 BPNGLoggerDetector.hh File Reference	87
7.3 IBPNGClient.h File Reference	88
7.4 IBPNGClientListener.h File Reference	90
7.5 IClientProperties.h File Reference	91
7.6 RdbDefines.h File Reference	92
7.7 RdbEventList.hh File Reference	93
Index	93

Kapitel 1

User's manual - Telemotive Client Library

3.3.1

1.1 General

This is the documentation for the C++ Telemotive 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 Telemotive Client Library provides methods for base functionality like:

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

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

1.2.1 Error handling and listener mechanism

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

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

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

1.3 Compiler/Linker

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

1.4 Thread safety

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

1.5 Demo project

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

Exampe for lib unsage:

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

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

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

// atom
#include "BPNGDefines.h"

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

#ifdef _MSC_VER
#include <sys/stat.h>
#else
```



```

#include <direct.h>
#define mkdir(a, b) _mkdir(a)
#endif

using namespace std;

/*****ONLINE*DOWNLOAD*****/
static void sampleFunctionDownload(OnlineLoggerInfo device);
static void sampleFunctionTSLDownload(TSLCluster tsl);
/*****ONLINE*CONVERSION*****/
static void sampleFunctionOnlineConversion(OnlineLoggerInfo device);
static void sampleFunctionTSLOnlineConversion(TSLCluster tsl);
/*****CONFIG*****/
static void sampleFunctionConfiguration(OnlineLoggerInfo device);
static void sampleFunctionTSLConfiguration(TSLCluster tsl);
/*****OFFLINE*CONVERSION*****/
static void sampleFunctionOfflineConversion();
static void sampleFunctionOfflineTSLConversion();
/*****HELPERS*****/
static string getLocalDateString();
static vector<string> readZipsFromDirectory(string dir);

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

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

    //map for tsl's

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

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

        tsl.print();

        cout << "\n";

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

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

    if (tslFound)

```



```

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

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

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

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

// We want to download all traces since last startup to an offline data set
static void sampleFunctionDownload(OnlineLoggerInfo device)
{
    IBPNGClient *client = getBPNGClient();

    // connect logger
    BOOL ret = client->connectLogger(1, &device);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect logger. " << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

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

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

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

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

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

    DataSpan span;
    span.type = DST_IDSPAN;
    span.start = startupId;
    span.end = endId;

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

    ret = mkdir("../testoutdir", 0x777);

```



```

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

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

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

// We want to download all traces since last startup to an offline data set
static void sampleFunctionTSLDownload(TSLCluster tsl)
{
    IBPNGClient *client = getTSLClient(tsl.getNumMembers(
        TSLCluster::DOWNLOAD));

    // connect logger
    vector<OnlineLoggerInfo> devices = tsl.getConnectionVector(
        TSLCluster::DOWNLOAD);
    BOOL ret = client->connectLogger(devices.size(), &devices[0]);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect tsl. " << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

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

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

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

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

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

```



```

DataSpan span;
span.type = DST_IDSPAN;
span.start = startupId;
span.end = endId;

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

ret = mkdir("../testoutdir", 0x777);
if (ret != 0 && errno != EEXIST)
{
    cout << "Failed to create output directory" << endl;
    client->disconnectLogger();
    client->release();
    return;
}

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

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

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

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

    // connect logger
    BOOL ret = client->connectLogger(1, &device);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect logger." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

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

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

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

```



```

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

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

// Ensure the out directory exists
ret = mkdir("../testoutdir", 0x777);
if (ret != 0 && errno != EEXIST)
{
    cout << "Failed to create output directory" << endl;
    client->disconnectLogger();
    client->release();
    return;
}

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

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

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

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

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

```



```

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

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

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

static void sampleFunctionTSLOnlineConversion(TSLCluster tsl)
{
    IBPNGClient *client = getTSLClient(tsl.getNumMembers(
        TSLCluster::CONVERSION));

    // connect logger
    vector<OnlineLoggerInfo> devices = tsl.getConnectionVector(
        TSLCluster::CONVERSION);
    BOOL ret = client->connectLogger(devices.size(), &devices[0]);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect tsl." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

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

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

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

    if (markerTimeStamp == 0)

```



```

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

// Ensure the out directory exists
ret = mkdir("../testoutdir", 0x777);
if (ret != 0 && errno != EEXIST)
{
    cout << "Failed to create output directory" << endl;
    client->disconnectLogger();
    client->release();
    return;
}

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

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

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

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

    // Note: channel indices are zero based
    int index = channels->getChannel(i)->getIndex();
    if (index == 0 || index == 1)
    {
        // CAN #1 and #2 in separate files
        // -1 as fileId parameter creates a separate file for this channel
        conversionSet->addChannel(type,
            index,
            CANOE,
            -1,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    else if (index == 2 || index == 3)
    {
        // CAN #3 and #4 in the same file.
        // fileId != -1 will write all channels with the same format and same
        // file Id to the same output file (if procurable in accordance with
        // the format specification.
        conversionSet->addChannel(type,
            index,
            CANOE,
            10,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
    else
    {
        // All other CAN channels to one BLF file.
        conversionSet->addChannel(type,
            index,
            BLF,
            20,
            channels->getChannel(i)->getOffset(),
            channels->getChannel(i)->getMainboardNumber(),
            channels->getChannel(i)->isMappingActive(),
            channels->getChannel(i)->getMappedChannelIndex());
    }
}

```



```

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

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

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

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

    BOOL ret = client->connectLogger(1, &device);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect logger." << endl;
        cout << "BPNGErrorCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    // save current config
    stringstream targetPath;
    targetPath << "..\\testoutdir\\bpng_[" << device.mbnr << "][" << getLocalDateString() << "].zip";
    ret = client->getConfig(targetPath.str().c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to download configuration." << endl;
        cout << "BPNGErrorCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    // here you could change the downloaded configuration
    // by extracting it and modifying the xml files
    // see documentation of IBPNGClient::getConfig()
    // or IBPNGClient::reconfigLogger().
    // the new config archive needs a date in its filename in this form: YYYY-MM-DD_HH-MM-SS

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

    // Setting the default config
    ret = client->setDefaultConfig();
    if (ret == 0)
    {
        BPNGError err = client->getLastError();

```



```

        cout << "Failed to set default config to the logger." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

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

// This function shows how to:
// - download the configuration
// - reconfigure the logger device
// - set the default config
static void sampleFunctionTSLConfiguration(TSLCluster tsl)
{
    //get the tsl client instance
    IBPNGClient *client = getTSLClient(tsl.getNumMembers(
        TSLCluster::CONFIG));

    vector<OnlineLoggerInfo> devices = tsl.getConnectionVector(
        TSLCluster::CONFIG);
    BOOL ret = client->connectLogger(devices.size(), &devices[0]);
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to connect TSL." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->release();
        return;
    }

    // save current config
    string targetPath = "..\\testoutdir\\\\tsl_[" + tsl.getTSLName() + "]-[" + getLocalDateString()
        + "].";
    ret = client->getConfig(targetPath.c_str());
    if (ret == 0)
    {
        BPNGError err = client->getLastError();
        cout << "Failed to download configurations." << endl;
        cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
        client->disconnectLogger();
        client->release();
        return;
    }

    // here you could change the downloaded configuration
    // by extracting it and modifying the xml files
    // see documentation of IBPNGClient::getConfig()
    // or IBPNGClient::reconfigLogger().
    // the new config archive needs a date in its filename in this form: YYYY-MM-DD_HH-MM-SS

    // We use the same config that we downloaded
    // for tsl we have to create one string which includes all config-ip pairs
    vector<string> configZips = readZipsFromDirectory(targetPath);

    // prepare string <configpath:1>|<ipAddress:1>;<configPath:2>|<ipAddress:2>;...
    stringstream newConfigPath;

    for (vector<OnlineLoggerInfo>::iterator iter = tsl.begin(); iter != tsl.
        end(); ++iter)
    {
        OnlineLoggerInfo device = *iter;
        for (size_t i = 0; i != configZips.size(); ++i)
        {
            string configZip = configZips[i];
            if (configZip.find(device.mbnr) != string::npos)
            {
                newConfigPath << targetPath << "\\\" << configZip << "|\" << device.
                ip << ";";
                break;
            }
        }
    }

    ret = client->reconfigLogger(newConfigPath.str().c_str());
    if (ret == 0)

```



```

{
    BPNGError err = client->getLastError();
    cout << "Failed to reconfigure the logger." << endl;
    cout << "BPNGErrCode: " << err.code << ", " << err.msg << endl;
    client->disconnectLogger();
    client->release();
    return;
}

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

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

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

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

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

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

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

    // Ensure the out directory exists
    ret = mkdir("../testoutdir", 0x777);
    if (ret != 0 && errno != EEXIST)
    {
        cout << "Failed to create output directory" << endl;
    }
}

```



```

        client->release();
        return;
    }

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

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

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

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

        // Note: channel indices are zero based
        int index = channels->getChannel(i)->getIndex();
        if (index == 0 || index == 1)
        {
            // CAN #1 and #2 in separate files
            // -1 as fileId parameter creates a separate file for this channel
            conversionSet->addChannel(type,
                index,
                CANOE,
                -1,
                channels->getChannel(i)->getOffset(),
                channels->getChannel(i)->getMainboardNumber(),
                channels->getChannel(i)->isMappingActive(),
                channels->getChannel(i)->getMappedChannelIndex());
        }
        else if (index == 2 || index == 3)
        {
            // CAN #3 and #4 in the same file.
            // fileId != -1 will write all channels with the same format and same
            // fileId to the same output file (if procurable in accordance with
            // the format specification.
            conversionSet->addChannel(type,
                index,
                CANOE,
                10,
                channels->getChannel(i)->getOffset(),
                channels->getChannel(i)->getMainboardNumber(),
                channels->getChannel(i)->isMappingActive(),
                channels->getChannel(i)->getMappedChannelIndex());
        }
        else
        {
            // All other CAN channels to one BLF file.
            conversionSet->addChannel(type,
                index,
                BLF,
                20,
                channels->getChannel(i)->getOffset(),
                channels->getChannel(i)->getMainboardNumber(),
                channels->getChannel(i)->isMappingActive(),
                channels->getChannel(i)->getMappedChannelIndex());
        }
    }

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

    // free memory

```



```

    client->release();
}

// We want to convert all A/I, D/I traces from an Offline TSL data set
// to XTMT
static void sampleFunctionOfflineTSLConversion()
{
    // We use the sample Offline Data Set that was downloaded
    // with sampleFunctionTSLDownload().
    // Its up to you to ensure an existing file.
    // informations about number of tsl members can be retrieved by client lib function
    getNumTSLMemberFromOfflineDataSet
    string offlineDataSetPath = "..\\testoutdir\\TSL_Offline.zip";
    int numMembers = 0;
    getNumTSLMemberFromOfflineDataSet(offlineDataSetPath.c_str(), &
        numMembers);

    // create a tsl client: parameter is number of tsl member
    IBPNGClient *client = getTSLClient(numMembers);

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

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

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

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

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

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

```



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

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

    return buffer;
}

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

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

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

        entry = readdir(directory);
    }

    closedir(directory);
    return output;
}
```


Kapitel 2

Deprecated List

Member `DEV_BP2`

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

Member `IBPNGClient::connectLogger (const char *ipAddress)=0`

use `connectLogger(int numLogger, OnlineLoggerInfo* devices)` instead

Member `IBPNGClient::createNewConversionSet ()=0`

, use static function `createNewConversionSet()` instead

Member `IBPNGClient::getLicenses ()=0`

use `getLicenses(unsigned deviceMbnr)` instead

Member `IBPNGClient::getVersions ()=0`

use `getVersions(OnlineLoggerInfoStringPair *versionPairs)` instead

Member `IBPNGClient::reconfigLogger (const char *configZip)=0`

use `reconfigLogger(int numLogger, OnlineLoggerInfoStringPair *loggerToConfigPathPairs)` instead

Member `IBPNGClient::updateFirmware (const char *fwPath, BOOL force)=0`

use `updateFirmware(OnlineLoggerInfoStringPair *loggerToFirmwareUpdatePacketPair, BOOL force)` instead

Member `IBPNGClient::updateLicenses (const char *licenseFilePath)=0`

use `updateLicenses(OnlineLoggerInfoStringPair *loggerLicenseFilePair)` instead

Kapitel 3

Hierarchical Index

3.1 Class Hierarchy

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

BPNGError	21
DataSpan	28
IBPNGClient	28
IBPNGClientListener	50
BPNGLoggerDetector	21
IChannel	55
IChannelList	56
IClientProperties	56
IConversionSet	62
IFalseMeasureSignal	64
IFalseMeasureSignalList	64
IFormatInfo	65
IFormatList	66
IRdbEvent	66
IRdbEventList	67
IRdbTraceBlock	67
IRdbTraceBlockList	68
ITesttoolsChannel	68
ITesttoolsChannelList	69
LogInData	70
MemoryFillLevel	70
OnlineLoggerInfo	71
OnlineLoggerInfoStringPair	72
RdbEvent2	73
TSLCluster	74
vector	
RdbEventList	74

Kapitel 4

Class Index

4.1 Class List

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

BPNGError	
Error struct with error code and optional error message	21
BPNGLoggerDetector	21
DataSpan	28
IBPNGClient	
Interface class for the Telemotive Client Library	28
IBPNGClientListener	50
IChannel	
Channel interface	55
IChannelList	
Channel list interface	56
IClientProperties	
The IClientProperties interface replaces the deprecated <i>ClientProperties</i> struct	56
IConversionSet	
A conversion set stores all conversion relevant settings	62
IFalseMeasureSignal	
False measure signal interface	64
IFalseMeasureSignalList	
False measure signal list interface	64
IFormatInfo	
FormatInfo interface	65
IFormatList	
Format list interface	66
IRdbEvent	
Interface to an RDB event	66
IRdbEventList	
Interface to a list of rdb events	67
IRdbTraceBlock	67
IRdbTraceBlockList	68
ITesttoolsChannel	
Channel interface	68
ITesttoolsChannelList	
TesttoolsChannel list interface	69

LoginData	
Structure for login	70
MemoryFillLevel	
Stores memory fill level of a device	70
OnlineLoggerInfo	
Struct with information about a logger found in LAN	71
OnlineLoggerInfoStringPair	
Helper object for configuration, license update or firmwareupdate: a key value pair for assigning a configuration, licensefile, etc. to a device	72
RdbEvent2	
Implementation class for a wrapper of IRdbEvent using STL classes	73
RdbEventList	
Implementation class for a wrapper of IRdbEventList using STL classes	74
TSLCluster	74

Kapitel 5

File Index

5.1 File List

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

BPNGDefines.h	Defines for Telemotive Client Library	78
BPNGLoggerDetector.hh	Logger Detector Sample	87
IBPNGClient.h	Interface class for the BPNGClient DLL	88
IBPNGClientListener.h	Interface class for the BPNGClient listener	90
IClientProperties.h	Interface for client properties	91
RdbDefines.h	Public interfaces for Telemotive Reference Database access	92
RdbEventList.hh	IRdbEvent wrapper	93

Kapitel 6

Class Documentation

6.1 BPNGError Struct Reference

Error struct with error code and optional error message.

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

Public Attributes

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

6.1.1 Detailed Description

Error struct with error code and optional error message.

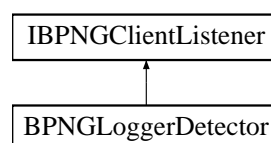
The documentation for this struct was generated from the following file:

- [BPNGDefines.h](#)

6.2 BPNGLoggerDetector Class Reference

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

Inheritance diagram for BPNGLoggerDetector:



Public Member Functions

- [BPNGLoggerDetector](#) ()
- `std::vector< OnlineLoggerInfo > getLoggerList` (unsigned searchTimeOut)
- `std::vector< TSLCluster > getTSLs` (`std::vector< OnlineLoggerInfo > loggersInNetwork`)
- virtual void WINAPI [onBPNGDeviceDetected](#) ([OnlineLoggerInfo](#) *info)
Called to notify a detected logger in network.
- virtual void WINAPI [onBPNGDeviceDisappeared](#) ([OnlineLoggerInfo](#) *info)
Called to notify a disappeared logger.
- virtual void WINAPI [onBPNGDeviceStateChange](#) ([OnlineLoggerInfo](#) *info)
Called to notify a logger's state change.
- virtual int WINAPI [onProgressDataDownload](#) (int percentCompleted)
Called to indicate the current progress of a file transfer.
- virtual int WINAPI [onProgressConversion](#) (int percentCompleted, const char *status)
Called to indicate the current progress of file conversion.
- virtual void WINAPI [onStatusMessage](#) (const char *statusMsg)
Called to send additional information of the current process to the calling app.
- virtual int WINAPI [onDataRecoverProgress](#) (const char *statusMsg, int percentage)
Called to send additional information of the current data recovery progress.
- virtual void WINAPI [onWarning](#) ([BPNGWarningCode](#) warningCode, const char *warnMsg)

Called to inform about a warning.
- virtual int WINAPI [onTargetPathTooLong](#) (char *newTarget, int maxSize)
Called on a too long target directory.
- virtual int WINAPI [getOverwritingPermission](#) (const char *filePath)
Called on existing output trace files.
- virtual const char *WINAPI [onLogInDataRequired](#) (unsigned mbnr)
Called on accessing password protected functions.
- virtual void WINAPI [onInvalidPwConfigFound](#) (unsigned mbnr)
Called if invalid pw file found on device.
- virtual void WINAPI [onLogInDataFailed](#) ()
- virtual void WINAPI [onResetLogInDataFailed](#) ()
- virtual void WINAPI [onFuncAccessDenied](#) ()
- virtual int WINAPI [onCriticalDiskSpace](#) (uint64_t freeSpace, uint64_t neededSpace, const char *drive)
- virtual void WINAPI [onFirmwareUpdateProgress](#) (int percentage, int stepId, int subStepId, const char *desc)
Called on firmware update progress.
- virtual void WINAPI [onFirmwareUpdateError](#) (int errorId)
- virtual int WINAPI [onGetLogReportProgress](#) (int percentage, const char *desc)
- virtual int WINAPI [onCriticalDiskSpace](#) (uint64_t freeSpace, uint64_t neededSpace, const char *drive, const char *msg)
Called in case of not enough free disk space.
- virtual void WINAPI [onDownloadStart](#) (int64_t totalAmountOfBytes)
Notifies the listeners before the download starts about the total amount of bytes to be downloaded.
- virtual void WINAPI [onConversionStart](#) (int64_t totalAmountOfBytes)
Notifies the listeners before the conversion starts about the total amount of bytes to be converted.
- virtual const char *WINAPI [onExtractionPasswordRequired](#) (uint8_t retryCount)

6.2.1 Detailed Description

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

6.2.2 Constructor & Destructor Documentation

6.2.2.1 BPNGLoggerDetector::BPNGLoggerDetector () [inline]

Constructor

6.2.3 Member Function Documentation

6.2.3.1 std::vector<OnlineLoggerInfo> BPNGLoggerDetector::getLoggerList (unsigned searchTimeout)

Returns a vector of detected BPNGDevice in local networks.

Parameters

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

Returns

vector of BPNGDevice

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

Called on existing output trace files.

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

Implements [IBPNGClientListener](#).

6.2.3.3 std::vector<TSLCluster> BPNGLoggerDetector::getTSLs (std::vector< OnlineLoggerInfo > loggersInNetwork)

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

Parameters

<i>loggersIn-Network</i>	the BPNGDevice in network found by getLoggerList(unsigned searchTimeout)
--------------------------	--

Returns

vector of [TSLCluster](#)

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

Called to notify a detected logger in network.

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

Implements [IBPNGClientListener](#).

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

Called to notify a disappeared logger.

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

Implements [IBPNGClientListener](#).

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

Called to notify a logger's state change.

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

Implements [IBPNGClientListener](#).

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

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

Parameters

<i>totalAmount-OfBytes</i>	Total data size to be converted
----------------------------	---------------------------------

Implements [IBPNGClientListener](#).

6.2.3.8 virtual int WINAPI BPNGLoggerDetector::onCriticalDiskSpace (uint64_t *freeSpace*, uint64_t *neededSpace*, const char * *drive*, const char * *msg*) [inline], [virtual]

Called in case of not enough free disk space.

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

Parameters

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

Returns

return 0 when process should be aborted, 1 to ignore

Implements [IBPNGClientListener](#).

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

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

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

Implements [IBPNGClientListener](#).

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

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

Parameters

<i>totalAmount-OfBytes</i>	Total data size to be downloaded
----------------------------	----------------------------------

Implements [IBPNGClientListener](#).

6.2.3.11 virtual const char* WINAPI BPNGLoggerDetector::onExtractionPasswordRequired (uint8_t *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](#).

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

Called on creation of log report

Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

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

Called if invalid pw file found on device.

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

Implements [IBPNGClientListener](#).

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

Called on accessing password protected functions.

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

Parameters

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

Implements [IBPNGClientListener](#).

6.2.3.15 `virtual int WINAPI BPNGLoggerDetector::onProgressConversion (int percentCompleted, const char * status)` `[inline], [virtual]`

Called to indicate the current progress of file conversion.

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

Parameters

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

Returns

return value 0 indicates an abort request from the implementing class

Implements [IBPNGClientListener](#).

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

Called to indicate the current progress of a file transfer.

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

Parameters

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

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

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

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

Implements [IBPNGClientListener](#).

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

Called on a too long target directory.

Called when the resulting file name of the converted files or the files of an offline data set is longer than the maximum allowed size of the file system (Windows 260). The lib user has to pass a new (shorter) base target directory to the passed char array with strcpy. The memory of the array is already allocated by the library and it's size is maxSize. When a new directory was set the value 1 must be returned. Returning another value than 1 will abort the current process with an error result.

Implements [IBPNGClientListener](#).

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

Called to inform about a warning.

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

Implements [IBPNGClientListener](#).

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

- [BPNGLoggerDetector.hh](#)

6.3 DataSpan Struct Reference

Public Attributes

- uint8_t [type](#)
set type to 0 for a id based range, set type to 1 for a time based range
- uint64_t [start](#)
start time/id of data span
- uint64_t [end](#)
end time/id of data span
- uint64_t **reserved**

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

- [BPNGDefines.h](#)

6.4 IBPNGClient Struct Reference

Interface class for the Telemotive Client Library.

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

Public Member Functions

- virtual void WINAPI [scanNetworkForLogger](#) ()=0
Scan network for logger.
- virtual void WINAPI [activateGatewayLoggerDetection](#) ()=0
Activates the detection of devices connected to a different subnet.
- virtual BOOL WINAPI [connectLogger](#) (const char *ipAddress)=0
Connect to logger with passed IP address.
- 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 BOOL WINAPI **initOnline** ()=0

Initialisation of download and online conversion process.
- virtual BOOL WINAPI **initOffline** (const char *path)=0

Initialisation of offline conversion process.
- virtual int WINAPI **downloadDataSpans** (uint16_t numSpans, **DataSpan** *dataSpans, const char *target, BOOL doSorting)=0

Download trace data.
- virtual **IConversionSet** *WINAPI **createNewConversionSet** ()=0

Returns the pointer to a new conversion set.
- virtual int WINAPI **convertData** (**IConversionSet** *conversionSet, const char *target)=0

Convert all data specified by conversionSet.
- virtual BOOL WINAPI **getConfig** (const char *path)=0

Download the current logger configuration to the passed path.
- virtual BOOL WINAPI **reconfigLogger** (const char *configZip)=0

Reconfig logger with the zipped new configuration.
- virtual BOOL WINAPI **reconfigLogger** (int numLogger, **OnlineLoggerInfoStringPair** *logger-ConfigPathPairs)=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 const char *WINAPI **getVersions** ()=0

Get the firmware and hardware version string.
- 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](#) (const char *fwPath, BOOL force)=0
Update firmware.
- virtual BOOL WINAPI [updateFirmware](#) (OnlineLoggerInfoStringPair *loggerFirmwareUpdatePacketPair, BOOL force)=0
Update firmware.
- virtual BOOL WINAPI [isUserAuthenticated](#) (PwdPrivilegesFuncId actionName)=0
- virtual BOOL WINAPI [updateLicenses](#) (const char *licenseFilePath)=0
Update licenses.
- virtual BOOL WINAPI [updateLicenses](#) (OnlineLoggerInfoStringPair *loggerLicenseFilePair)=0
Update licenses.
- virtual const char *WINAPI [getLicenses](#) ()=0
Returns the license file's content as string.
- 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 [deleteData](#) (uint16_t numSpans, [DataSpan](#) *dataSpans)=0
Delete trace data.
- virtual int WINAPI [deleteAllData](#) ()=0
Delete all trace data on the logger.
- virtual BOOL WINAPI [setInfoEvent](#) (const char *msg)=0
Set an info event with the passed string on the connected logger.
- virtual BOOL WINAPI [setMarker](#) ()=0
Set a marker on the connected logger. Returns 0 on error.
- virtual int WINAPI [getCurrentLoggerTime](#) ()=0
Returns the current loggertime in seconds since 01.01.1970 UTC.
- virtual int WINAPI [setTime](#) (int time)=0
Set logger time and date to the passed UTC time stamp.
- virtual void WINAPI [keepLoggerAlive](#) (const char *ip)=0
Call this to keep logger alive.
- virtual void WINAPI [stopKeepLoggerAlive](#) (const char *ip)=0
Called to stop sending keep alive pings to the logger specified via the passed ip.
- virtual [OnlineLoggerInfo](#) *WINAPI [getOnlineLoggerInfo](#) (const char *ip)=0
Retreive an [OnlineLoggerInfo](#) for a particular IP address.
- virtual [IFormatList](#) *WINAPI [getAvailableFormats](#) ()=0
Return pointer to a format list interface. Returns null in case of error.
- virtual void WINAPI [flashDeviceLED](#) ()=0
Let the connected device blink its front LEDs for identification.
- virtual int WINAPI [createCCPXCPSeqFile](#) (const char *xsdDir, const char *xmlDir, bool forceFlag)=0
Parse CCPXCPMeasurement.xml and CCPXCPCConfiguration.xml and create CCPXCPSquence.xml.
- virtual int WINAPI [createCCPXCPDbcFiles](#) (const char *dbcDir, const char *xsdDir, const char *xmlDir)=0

Parse CCPXCPMeasurement.xml and 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 [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 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

6.4.1 Detailed Description

Interface class for the Telemotive Client Library.

[IBPNGClient](#) is the interface class of the blue PiraT Client library. To get access to a single blue PiraT 2, blue PiraT 2 5E, blue PiraT mini or blue PiraT Remote data logger you need a pointer to an implementing instance of the [IBPNGClient](#) interface. Use [getBPNGClient\(\)](#) to get such a pointer. This will create an implementing instance on the heap. To avoid conflict between different runtime libraries it is obligatory to release this object with its [IBPNGClient::release\(\)](#) function when not needed any more. Don't call the delete operator directly on this pointer.

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. Use [getTSLClient\(int numTSL-Member\)](#) to get such a pointer. It must also be released with [IBPNGClient::release\(\)](#).

6.4.2 Member Function Documentation

6.4.2.1 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.

6.4.2.2 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.

Parameters

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

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

Connect to logger with passed IP address.

Deprecated use [connectLogger\(int numLogger, OnlineLoggerInfo* devices\)](#) instead

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

On TSL instance you have to pass all ip addresses of devices in TSL. The ip's have to be separated by semicolons ';'. For example: "192.168.0.233;192.168.0.10;192.168.0.1". The number of ip's have to match the number on [getTSLClient\(int numTSLMember\)](#) creation, else the connect function will return with a failure.

cannot be used to connect to a WLAN-TSL

will be removed with next vesion!

Parameters

<i>ipAddress</i>	IP address of the logger that should be connected
------------------	---

Returns

0 on failure, 1 on success

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

Connect to passed loggers.

While the loggers are connected, they won't go to standby mode until the last [IBPNGClient](#) instance is disconnected. If connect fails the function will return 0. On success the return value is 1. In case of failure further information can be retrieved with [getLastError\(\)](#).

Parameters

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

Returns

0 on failure, 1 on success

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

Convert all data specified by conversionSet.

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

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

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

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

Parameters

<i>conversionSet</i>	conversion settings, see IConversionSet
<i>target</i>	target directory for the converted trace files. Depending on the passed Client-Properties the files may be stored in sub folders named by date

Returns

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

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

Returns the pointer to a new conversion set.

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

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

easy track test report creation

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

Parameters

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

Returns

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

6.4.2.8 `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.

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

Delete trace data.

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

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

Parameters

<i>numSpans</i>	Size of the passed DataSpan array in second parameter
<i>dataSpans</i>	Array of DataSpan , specifying the time or ID spans that should be deleted

Returns

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

6.4.2.10 `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_TI-MESPAN_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_TI-MESPAN_TRACES

Returns

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

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

Download trace data.

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

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

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

Parameters

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

Returns

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

6.4.2.12 `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 [initOffline](#) and filters signals according to the complex filter settings created with the Telemotive System Client.

Parameters

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

6.4.2.13 `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 Telemotive System Client.

Parameters

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

6.4.2.14 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.

6.4.2.15 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](#)

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

See Also

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

6.4.2.17 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
-------------	--

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

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

After calling one of the init functions [IBPNGClient::initOnline\(\)](#) or [IBPNGClient::initOffline\(\)](#) 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:...\Telemotive_bp2\PNGINST25452\B-PTSL_10.64.76.171_2\BP2Img_10.64.76.171_3_PID5452;C:...\Telemotive_bp2\BPNGINST25452\B-PTSL_10.64.76.171_2\BP2Img_10.64.76.206_4_PID5452;`

Returns

Path to the folder containing the extracted config archive.

6.4.2.19 `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.

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

Get name of device.

After calling one of the init functions [IBPNGClient::initOnline\(\)](#) or [IBPNGClient::initOffline\(\)](#) this function returns the currently configured device name. On TSL the device names will be separated by ','

Returns

The device name

See Also

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

6.4.2.21 `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.

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

Get list of all events from the RDB.

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

Returns

Pointer to a [IRdbEventList](#)

6.4.2.23 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](#)

6.4.2.24 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](#)

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

Returns the license file's content as string.

Deprecated use [getLicenses\(unsigned deviceMbnr\)](#) instead

On TSL instance you get one string with licenses of all participants separated by '|' <mainboardnumber:1>:<license:1>|<mainboardnumber:2>:<license:2> /

will be removed with next vesion!

6.4.2.26 `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

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

Returns pointer to a channel list interface.

After calling one of the init functions [IBPNGClient::initOnline\(\)](#) or [IBPNGClient::initOffline\(\)](#) this function returns a pointer to the logger's/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](#)

6.4.2.28 `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

6.4.2.29 `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\(\)](#)

6.4.2.30 `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\(\)](#)

6.4.2.31 `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

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

Get path to the reference data base.

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

Returns

Path to the downloaded or extracted RDB file

See Also

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

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

Get list of all trace blocks from the RDB.

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

Returns

Pointer to a [IRdbEventList](#)

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

Get the firmware and hardware version string.

Deprecated use [getVersions\(OnlineLoggerInfoStringPair *versionPairs\)](#) instead

On TSL the result will be splitted by "<versionstring>|<mainboardnumber>\n"

will be removed with next vesion!

Returns

the firmware and hardware version string

6.4.2.35 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

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

Initialisation of offline conversion process.

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

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

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

If you have an TSL offline data set ensure you created the IBPNGClientLib instance by calling [getTSLClient\(int numTSLMember\)](#). The numTSLMember can be retrieved by calling [getNumTSLMemberFromOfflineDataSet\(const char* offlinePath, int* numMember\)](#). The T-SL offline data set can be identified by the abstract_TSL.txt in the base path.

Returns

0 on failure, 1 on success

6.4.2.37 virtual BOOL WINAPI IBPNGClient::initOnline () [pure virtual]

Initialisation of download and online conversion process.

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

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

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

Returns

0 on failure, 1 on success

6.4.2.38 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

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

Call this to keep logger alive.

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

Parameters

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

See Also

[stopKeepLoggerAlive\(\)](#)

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

Reconfig logger with the zipped new configuration.

Deprecated use [reconfigLogger\(int numLogger, OnlineLoggerInfoStringPair *loggerToConfigPathPairs\)](#) instead

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

On TSL instance you have to pass all configurations in one string using this format:

<configpath:1>|<ipAddress:1>;<configPath:2>|<ipAddress:2>;... for example ..\testoutdir\tsl-[emaNLSToN]_[2016-05-02_18-03-13]\BP2Img_MBNR_1014179.zip|10.64.76.202;..\testoutdir\tsl-[emaNLSToN]_[2016-05-02_18-03-13]\BP2Img_MBNR_1026651.zip|10.64.76.149;

cannot be used to reconfigure a WLAN-TSL

will be removed with next vesion!

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 specifiy all xml files that are mandatory to reconfigure the data logger. You can validate the xml files with the supplied xsd files and a XML library of your choice. One possibility would be the XERCES library, see <http://xerces.apache.org/xerces-c/>

Parameters

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

Returns

0 on failure, 1 on success

6.4.2.41 virtual BOOL WINAPI IBPNGClient::reconfigLogger (int *numLogger*,
OnlineLoggerInfoStringPair * *loggerConfigPathPairs*) [pure virtual]

Reconfig logger with the zipped new configuration.

Reconfigurates the logger/tsl with the passed configurations. The ZIP archive can be either one that was downloaded with the [getConfig\(\)](#) method, stored by the client software or a modified

one. If you want to create your own configuration ZIP archive the structure of this file must be the same as of those mentioned above (xml files inside an "etc" directory). The abstract.txt file and all *.xsd files are optional. The filename must include the current date in followed form: [YYYY-MM-DD_HH-MM-SS] -> Y=year, M=month, D=day, H=hour, M=minute, S=second

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

See Also

[OnlineLoggerInfoStringPair](#)

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

Parameters

<i>numLogger</i>	Number of following OnlineLoggerInfoStringPair (should be equal to the number of devices on TSL)
<i>loggerTo-ConfigPath-Pairs</i>	Pointer to first OnlineLoggerInfoStringPair

Returns

0 on failure, 1 on success

6.4.2.42 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!

6.4.2.43 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

6.4.2.44 `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

6.4.2.45 `virtual void WINAPI IBPNGClient::scanNetworkForLogger ()` [pure virtual]

Scan network for logger.

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

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

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

Parameters

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

See Also

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

6.4.2.47 `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

6.4.2.48 `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

6.4.2.49 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

6.4.2.50 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

6.4.2.51 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

6.4.2.52 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

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

Synchronizes the RDB.

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

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

Update firmware.

Deprecated use [updateFirmware\(OnlineLoggerInfoStringPair *loggerToFirmwareUpdatePacketPair, BOOL force\)](#) instead

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.

On TSL instance you have to call a updateFirmware for every device to handle the update in multiple threads. The fwPath argument contains the firmwarepacket-path and the ip separately by '|' <firmwarepacket-path>|<ip> for example C:\bPMini_SW02-03-01.dat|10.64.-76.202

cannot be used to update loggers on a WLAN-TSL

will be removed with next vesion!

Parameters

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

Returns

0 on failure, 1 on success

6.4.2.55 virtual BOOL WINAPI IBPNGClient::updateFirmware (OnlineLoggerInfoStringPair * *loggerToFirmwareUpdatePacketPair*, 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= OnlineLoggerInfo , the device to be updated and value=local path to firmware paket
<i>force</i>	Flag whether to update the components independently from the components' versions

Returns

0 on failure, 1 on success

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

Update licenses.

Deprecated use [updateLicenses\(OnlineLoggerInfoStringPair *loggerLicenseFilePair\)](#) instead

Overwrites the current license file with the new one.

On TSL instance you have to call a updateLicenses for every device The licenseFilePath argument contains the target ip and the license-file-path separately by '|' <ip>|<license-FilePath> for example 10.64.76.202|C:\proj\Lizenz_bluePiraT2_MB_1014179.tml

cannot be used to update license files on loggers of a WLAN-TSL

will be removed with next vesion!

Parameters

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

Returns

0 on failure, 1 on success

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

Update licenses.

Overwrites the current license file with the new one.

Parameters

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

Returns

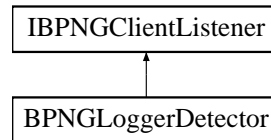
0 on failure, 1 on success

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 file transfer.
- virtual int WINAPI [onProgressConversion](#) (int percentCompleted, const char *status)=0
Called to indicate the current progress of file conversion.
- virtual void WINAPI [onStatusMessage](#) (const char *statusMsg)=0
Called to send additional information of the current process to the calling app.
- virtual 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](#) (uint8_t retryCount)=0

6.5.1 Member Function Documentation

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

Called on existing output trace files.

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

Implemented in [BPNGLoggerDetector](#).

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

Called to notify a detected logger in network.

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

Implemented in [BPNGLoggerDetector](#).

6.5.1.3 virtual void WINAPI IBPNGClientListener::onBPNGDeviceDisappeared ([OnlineLoggerInfo](#) * *info*) [pure virtual]

Called to notify a disappeared logger.

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

Implemented in [BPNGLoggerDetector](#).

6.5.1.4 virtual void WINAPI IBPNGClientListener::onBPNGDeviceStateChange ([OnlineLoggerInfo](#) * *info*) [pure virtual]

Called to notify a logger's state change.

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

Implemented in [BPNGLoggerDetector](#).

6.5.1.5 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>totalAmount-OfBytes</i>	Total data size to be converted
----------------------------	---------------------------------

Implemented in [BPNGLoggerDetector](#).

6.5.1.6 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](#).

6.5.1.7 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](#).

6.5.1.8 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>totalAmount-OfBytes</i>	Total data size to be downloaded
----------------------------	----------------------------------

Implemented in [BPNGLoggerDetector](#).

6.5.1.9 `virtual const char* WINAPI IBPNGClientListener::onExtractionPasswordRequired (uint8_t 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](#).

6.5.1.10 `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](#).

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

Called if invalid pw file found on device.

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

Implemented in [BPNGLoggerDetector](#).

6.5.1.12 `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
------------------	---

Implemented in [BPNGLoggerDetector](#).

6.5.1.13 `virtual int WINAPI IBPNGClientListener::onProgressConversion (int percentCompleted, const char * status)` [pure virtual]

Called to indicate the current progress of file conversion.

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

Parameters

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

Returns

return value 0 indicates an abort request from the implementing class

Implemented in [BPNGLoggerDetector](#).

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

Called to indicate the current progress of a file transfer.

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

Parameters

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

6.5.1.15 `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](#).

6.5.1.16 `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](#).

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

Called to inform about a warning.

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

Implemented in [BPNGLoggerDetector](#).

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

- [IBPNGClientListener.h](#)

6.6 IChannel Struct Reference

Channel interface.

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

Public Member Functions

- virtual [ChannelType](#) [getType](#) () const =0
Returns the ChannelType.
- virtual uint8_t [getIndex](#) () const =0
Returns the channel's index.
- virtual const char * [getName](#) () const =0
Returns the channel's name.
- virtual uint32_t [getMainboardNumber](#) () const =0
Returns the channel's package id.
- virtual uint32_t [getOffset](#) () const =0
Returns the channel's offset.
- virtual BOOL [isMappingActive](#) () const =0

Returns whether the channel is mapped.

- virtual uint8_t [getMappedChannelIndex](#) () const =0

Returns the channel's mapped channel index.

6.6.1 Detailed Description

Channel interface.

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

- [BPNGDefines.h](#)

6.7 IChannelList Struct Reference

Channel list interface.

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

Public Member Functions

- virtual int [getSize](#) () const =0

Returns the number of channels.

- virtual const [IChannel](#) * [getChannel](#) (int index) const =0

Returns the [IChannel](#) at index.

6.7.1 Detailed Description

Channel list interface.

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

- [BPNGDefines.h](#)

6.8 IClientProperties Struct Reference

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

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

Public Member Functions

- virtual void WINAPI [setCommonProperties](#) (const char *nameOfTester, int maxOutputSizeMB, BOOL separatedTimeFormat, BOOL separatedTimeFormatInOfflineSet, const char *alternativeLoggerName, BOOL useAlternativeLoggerName, BOOL useSubDirectories, BOOL midnightSplitting, BOOL fileTimeSpansLikeSelection, BOOL markerNumberInFileNames, BOOL subfolderWithLoggerName, int maxOfflineZipSizeMB, int maxOutputSizeMBSortedDownload, BOOL traceCutterStorage, const char *traceCutterFNPatternConversion, BOOL

createOfflineDataOnTraceCutterStorage, const char *traceCutterFNPatternOfflineData)=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 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 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 [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.

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

6.8.2.1 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

6.8.2.2 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

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

Set CAN pseudo properties for writing trigger messages.

Parameters

<i>writeTriggerMessage</i>	Active flag for writing CAN pseudo messages with trigger information
<i>channelIndex</i>	CAN channel for the trigger pseudo messages
<i>dlc</i>	DLC for the trigger pseudo messages
<i>canID</i>	CAN ID for the trigger pseudo messages

<i>triggerNum- BitPos</i>	Bit position for the trigger's index (16 bit length)
-------------------------------	--

6.8.2.4 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*) [pure virtual]

Set Common properties.

Parameters

<i>nameOfTester</i>	Name of tester that is written to the converted file names
<i>maxOutput- SizeMB</i>	Maximum file size for converted files. When this size is reached a new file is created.
<i>separated- TimeFormat</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>separated- TimeFormat- InOfflineSet</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>alternative- LoggerName</i>	The logger device's name is included in the converted files' names. An alternative logger name can be used.
<i>use- Alternative- LoggerName</i>	Set this field to 1 if the alternative logger name should be used in converted file names, 0 if not.
<i>useSub- Directories</i>	Set to 1 if converted files should be stored in subdirectories named by their start date, set 0 if they should not.
<i>midnight- Splitting</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>fileTime- SpansLike- Selection</i>	The file names of the converted files contain the time span of the included data. Setting this parameter to 1 will create time spans like they were specified in the IConversionSet . Setting this to 0 will create time spans according to the effectively included data.
<i>marker- NumberInFile- Names</i>	Specifies whether the indices of the marker included in a converted file should be appended to its file name
<i>subfolder- WithLogger- Name</i>	Specifies whether the name of the subfolder the converted files are stored in should contain the logger name or not.
<i>maxOutput- SizeMB- Sorted- Download</i>	Maximum file size for sorted download trace files. When this size is reached a new file is created.

6.8.2.5 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 [loadFormats](#) (const char *pathToIniFile)=0
Loads the format settings from an ini file.
- virtual bool [saveFormats](#) (const char *pathToIniFile)=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

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

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

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

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

Parameters

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

6.9.2.2 virtual void IConversionSet::addRdbIdRange (uint64_t *startId*, uint64_t *endId*) [pure virtual]

Adds a ReferenceDB ID range to the conversion set.

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

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

Parameters

<i>startId</i>	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

6.9.2.3 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

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 [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

6.14.2.1 `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.

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

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

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

Returns the unique entry ID that can be set to DataSpans for data download and conversion.

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

- [RdbDefines.h](#)

6.15 IRdbEventList Struct Reference

Interface to a list of rdb events.

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

Public Member Functions

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

6.15.1 Detailed Description

Interface to a list of rdb events.

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

- [RdbDefines.h](#)

6.16 IRdbTraceBlock Struct Reference

Public Member Functions

- `virtual 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)=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 GBytes
- uint8_t [percentageFill](#)
percentage filled
- uint8_t [percentageFillProtected](#)
percentage filled of protected areas
- uint32_t [extRingBufferSize](#)
size of external media ringbuffer in GBytes
- uint8_t [extPercentageFill](#)
external media percentage filled
- uint8_t [extPercentageFillProtected](#)
external media percentage filled of protected areas
- uint64_t [mbnr](#)
mainboardnumber of device

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.

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

Public Attributes

- const char * [ip](#)
the logger's ip address
- const char * [name](#)
the logger's name
- const char * [mbnr](#)
mainboard number
- 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

6.22.1 Detailed Description

Struct with information about a logger found in LAN.

6.22.2 Member Data Documentation

6.22.2.1 uint8_t OnlineLoggerInfo::deviceType

Device type,.

See Also

[BPNGDeviceType](#)

6.22.2.2 uint8_t OnlineLoggerInfo::loggerStatus

current logger status,

See Also

[BPNGLoggerStatus](#)

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

- [BPNGDefines.h](#)

6.23 OnlineLoggerInfoStringPair Struct Reference

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

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

Public Attributes

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

6.23.1 Detailed Description

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

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

- [BPNGDefines.h](#)

6.24 RdbEvent2 Struct Reference

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

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

Public Member Functions

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

Public Attributes

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

6.24.1 Detailed Description

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

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

See Also

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

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

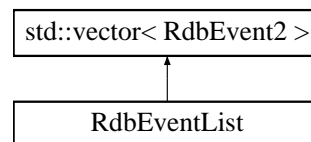
- [RdbEventList.hh](#)

6.25 RdbEventList Class Reference

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

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

Inheritance diagram for RdbEventList:



Public Member Functions

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

6.25.1 Detailed Description

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

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

See Also

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

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

- [RdbEventList.hh](#)

6.26 TSLCluster Class Reference

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

Public Types

- enum [ConnectionType](#) {
 [DOWNLOAD](#), [CONVERSION](#), [CONFIG](#), [BUGREPORT](#),
 [FW_UPDATE](#) }

Public Member Functions

- [TSLCluster](#) ()
- [TSLCluster](#) ([OnlineLoggerInfo](#) firstDevice)
- void [addDevice](#) ([OnlineLoggerInfo](#) device)
- std::vector< [OnlineLoggerInfo](#) > [getConnectionVector](#) ([ConnectionType](#) connectionType)
- int [getNumMembers](#) ([ConnectionType](#) connectionType)
- std::string [getTSLName](#) ()
- void [print](#) ()
- std::vector< [OnlineLoggerInfo](#) >::iterator [begin](#) ()
- std::vector< [OnlineLoggerInfo](#) >::iterator [end](#) ()

6.26.1 Detailed Description

A simple class that represents a Telemotive System Link chain.

6.26.2 Member Enumeration Documentation

6.26.2.1 enum TSLCluster::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

6.26.3 Constructor & Destructor Documentation

6.26.3.1 TSLCluster::TSLCluster () [inline]

Constructor

6.26.3.2 TSLCluster::TSLCluster ([OnlineLoggerInfo](#) firstDevice) [inline]

Constructor

Parameters

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

6.26.4 Member Function Documentation

6.26.4.1 `void TSLCluster::addDevice (OnlineLoggerInfo device)` `[inline]`

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

Parameters

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

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

Begin iterator for ranged base for loop

Returns

the begin iterator of internal BPNGDevice vector.

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

End iterator for ranged base for loop

Returns

the end iterator of internal BPNGDevice vector.

6.26.4.4 `std::vector<OnlineLoggerInfo> TSLCluster::getConnectionVector (ConnectionType connectionType)` `[inline]`

get the device list (ips separated by ';') for [IBPNGClient::connectLogger\(int numLogger, \[OnlineLoggerInfo* devices\]\(#\)\)](#)

It is recommended to exclude the Remote Control Touch on Download and Conversion cause its not a data logging device.

Parameters

<i>connection-Type</i>	the type of work for which the connection will be used
------------------------	--

Returns

the ip string

6.26.4.5 `int TSLCluster::getNumMembers (ConnectionType connectionType)` `[inline]`

get the number of participants for [IBPNGClient::getTSLClient\(int numTSLMember\)](#)

It is recommended to exclude the Remote Control Touch on Download and Conversion cause its not a data logging device.

Parameters

<i>connection-Type</i>	the type of work for which the connection will be used
------------------------	--

Returns

number of participants

6.26.4.6 `std::string TSLCluster::getTSLName () [inline]`

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

Returns

the TSL name.

6.26.4.7 `void TSLCluster::print () [inline]`

Stream the [TSLCluster](#) to cout.

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

- [BPNGLoggerDetector.hh](#)

Kapitel 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.
- struct [DataSpan](#)
- struct [BPNGError](#)
Error struct with error code and optional error message.

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

Macros

- #define **__BPNGDEFINES_H__**
- #define **WINAPI**
- #define **DECLDIR**
- #define **BOOL** bool
- #define **VOID** void

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_IP](#) = 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_ERROR](#) = 28, [BPNG_INITIALISATION_ERROR](#) = 29,
[BPNG_RDB_SQLITE_QUERY_ERROR](#) = 30, [BPNG_RDB_OPEN_FAILED](#) = 31, [BPNG_CONVERSION_ERRORS](#) = 32, [BPNG_CONV_SET_NOT_FOUND](#) = 33,
[BPNG_NOTHING_TO_CONVERT](#) = 34, [BPNG_TMT_FILE_ID_ERROR](#) = 35, [BPNG_TMT_FORMAT_ERROR_VERSION](#) = 36, [BPNG_TMT_FORMAT_ERROR_TS](#) = 37,
[BPNG_INVALID_MESSAGE_ERROR](#) = 38, [BPNG_INVALID_MESSAGE_ID](#) = 39, [BPNG-](#)


```

_BPNG_INVALID_MESSAGE_TS = 40, BPNG_INVALID_MESSAGE_SUBID = 41,
BPNG_INVALID_MESSAGE_LEN = 42, BPNG_CONV_FORMAT_ERROR = 43, BPNG_
_DOWNLOAD_ERRORS = 44, BPNG_NOTHING_TO_DOWNLOAD = 45,
BPNG_INVALID_OFFLINE_SET = 46, BPNG_PARAMETER_MISMATCH = 47, BPNG_F
_W_VERSION_CHECK_ERROR = 48, BPNG_USER_CANCELLED = 49,
BPNG_MIN_VERSION_ERROR = 50, BPNG_EXCEPTION = 51, BPNG_INCOMPATIBL
_E_RDB = 52, BPNG_UNSPECIFIED_ERROR = 53,
BPNG_LOAD_DBC_FAILED = 81, BPNG_CCP_XCP_PARSER_ERROR = 54, BPNG_
_CCP_XCP_DBC_GENERATOR_ERROR = 55, BPNG_CCP_XCP_SEQUENCE_GENE
RATOR_ERROR = 56,
BPNG_INSUFFICIENT_DISK_SPACE = 57, BPNG_FWUPDATE_FAILED = 58, BPNG_
_INDEX_OUT_OF_RANGE_ERR = 59, BPNG_READ_CONFIG_BACKUP_ERR = 60,
BPNG_INVALID_RPC_COMMAND = 61, BPNG_INVALID_TSL_CASCDING = 62, BPNG_
_LOGIN_CANCELED = 63, BPNG_USER_PWD_WRONG = 64,
BPNG_NO_ACCESS_FOR_FUNCTION = 65, BPNG_STREAMING_PROTOCOLL_ERR
OR = 68, BPNG_STREAMING_SOCKET_ERROR = 69, BPNG_STREAMING_DISABL
ED = 70,
BPNG_FW_DEPRECATED = 71, BPNG_STREAMING_ABORTED_BY_PEER = 72, BP
NG_INCONSISTENT_TSL_FWVERSIONS = 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_OFFLINEDATASET = 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 }

```

enum Error codes

- enum **FWUpdateErrorCode** {


```

FWUPDATE_ERRORCODE_NO_ERR = 0, FWUPDATE_ERRORCODE_FW_PKT_NA
ME_EMPTY = -2, FWUPDATE_ERRORCODE_FW_PKT_MISSING = -3, FWUPDATE_
_ERRORCODE_NAMED_PIPE_SERVER_MKNOD = -4,
FWUPDATE_ERRORCODE_NAMED_PIPE_SERVER_OPEN = -5, FWUPDATE_ERRO
RCODE_FW_UPDATE_NOT_IN_PROGRESS_TIMEOUT = -6, FWUPDATE_ERRORCO
DE_MISSING_LINUX_DISTR = -7, FWUPDATE_ERRORCODE_MISSING_LIBTMLIB_F
ILE = -8,
FWUPDATE_ERRORCODE_MISSING_TMLIB_FILE = -9, FWUPDATE_ERRORCODE_
MISSING_ATOM_FILE = -10, FWUPDATE_ERRORCODE_MISSING_CLIENT_FILE = -
11, FWUPDATE_ERRORCODE_MISSING_FPGAA_FILE = -12,
FWUPDATE_ERRORCODE_MISSING_FPGAB_FILE = -13, FWUPDATE_ERRORCOD
E_MISSING_EXTENSION_BOARD_FPGA_FILE = -14, FWUPDATE_ERRORCODE_MI
SSING_GBE_FILE = -15, FWUPDATE_ERRORCODE_MISSING_SBC_FILE = -16,
FWUPDATE_ERRORCODE_MISSING_SBC_FLASH_SCRIPT = -17, FWUPDATE_ERR
ORCODE_MISSING_FPGA_FLASH_SCRIPT = -18, FWUPDATE_ERRORCODE_MISSI
NG_RCV_FILE = -19, FWUPDATE_ERRORCODE_MISSING_LINUX_SETUP_ARCHIVE
= -20,
FWUPDATE_ERRORCODE_UNKNOWN_MB_HW_VERSION = -21, FWUPDATE_ERR
ORCODE_UNKNOWN_EXTENSION_BOARD = -22, FWUPDATE_ERRORCODE_UNK
KNOWN_EXTENSION_BOARD_VARIANCE = -23, FWUPDATE_ERRORCODE_NOT_R
EADABLE_EXTENSION_BOARD_VARIANCE = -24,
FWUPDATE_ERRORCODE_NOT_READABLE_EXTENSION_BOARD_HW_VERSION =
-25, FWUPDATE_ERRORCODE_NOT_READABLE_HW_TYPE_VERSION = -26, FWU
PDATE_ERRORCODE_FAILED_UPDATE_APP_LIBS = -27, FWUPDATE_ERRORCO
DE_FAILED_UPDATE_RC = -28,
FWUPDATE_ERRORCODE_FAILED_UPDATE_GBEC = -29, FWUPDATE_ERRORCO

```


- DE_CONV_CFG_ERROR** = -30, **FWUPDATE_ERRORCODE_FAILED_UNCOMPRESS_LINUX_KERNEL** = -31, **FWUPDATE_ERRORCODE_FAILED_UNCOMPRESS_LINUX_KERNEL_MODULES** = -32,
FWUPDATE_ERRORCODE_FAILED_CPY_LINUX_KERNEL = -33, **FWUPDATE_ERRORCODE_FAILED_UNCOMPRESS_CLIENT_FILE** = -34, **FWUPDATE_ERRORCODE_FAILED_CPY_CLIENT_FILE** = -35, **FWUPDATE_ERRORCODE_FAILED_UPDATE_LINUX_DISTR** = -36,
FWUPDATE_ERRORCODE_FAILED_SBC_FLASH = -37, **FWUPDATE_ERRORCODE_FAILED_UPDATE_CCP_XCP** = -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, **FWUPDATE_ERRORCODE_MISSING_SPYNIC_FILE** = -42, **FWUPDATE_ERRORCODE_MISSING_LOADING_ISPVM** = -43, **FWUPDATE_ERRORCODE_MISSING_DEVICE_FPGAB_FILE** = -44,
FWUPDATE_ERRORCODE_MISSING_DEVICE_FPGAA_FILE = -45, **FWUPDATE_ERRORCODE_UNREADY_FPGAA** = -46, **FWUPDATE_ERRORCODE_LINUX_KERNEL_MAY_FREEZE_SYSTEM** = -47, **FWUPDATE_ERRORCODE_NOT_SET_DEVICE_PATH** = -48,
FWUPDATE_ERRORCODE_NOT_SET_FW_FILE = -49, **FWUPDATE_ERRORCODE_NOT_SET_FPGA_KEY** = -50, **FWUPDATE_ERRORCODE_MISSING_DEVICE_FILE** = -51, **FWUPDATE_ERRORCODE_MISSING_FPGA_FW_FILE** = -52,
FWUPDATE_ERRORCODE_MISSING_TMUDEVQ = -53, **FWUPDATE_ERRORCODE_UNKNOWN_DEVICE_PATH** = -54, **FWUPDATE_ERRORCODE_FAILED_CPY_FPGA_FILE** = -55, **FWUPDATE_ERRORCODE_FAILED_LINKING_FW_UPDATE** = -56,
FWUPDATE_ERRORCODE_ERROR_FLASH_FPGA = -57, **FWUPDATE_ERRORCODE_ERROR_FLASH_SPYNIC** = -58, **FWUPDATE_ERRORCODE_ERROR_LOADING_FPGA_JTAG_DRIVER** = -59, **FWUPDATE_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_CONVERTED**, **BPNG_WARNING_NO_ESO_TRACE**,
BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONES, **BPNG_WARNING_RECOVERING_FAILED** }
Warning codes.
 - enum **LanguageID** { **BPNG_GERMAN**, **BPNG_ENGLISH** }
Languages.
 - enum **BPNGBugreportMode** {
BR_FULL_WO_TRACES = 0, **BR_ONLY_LOGS** = 1, **BR_FDB_RDB** = 2, **BR_ONLY_CLIENT** = 3,
BR_FULL_ALL_TRACES = 4, **BR_FULL_TIMESPAN_TRACES** = 5 }
Mode for the IBPNG::downloadBugReport() function.
 - enum **ChannelType** {
CH_UNDEFINED = 0, **OBSOLETE_CH_CANLS**, **CH_CAN**, **CH_LIN**,
CH_SERIAL, **CH_ETHERNET**, **CH_FLEXRAY**, **CH_MOST25_CTRL**,
CH_MOST25_MDP, **CH_MOST25_SYNC**, **CH_MOST150_CTRL**, **CH_MOST150_MDP**,
CH_MOST150_MEP, **CH_MOST150_STREAM**, **CH_ANALOG_IN**, **CH_DIGITAL_IN**,
CH_CAMERA, **CH_CCPXCP**, **CH_DIAG**, **CH_GPS**,
CH_ECL, **CH_COMPLEXFILTER**, **CH_TTY** }
Currently supported interfaces.

- enum **PwdPrivilegesFuncId** {
REMOVE_DATA = 0, **SET_TIME**, **SET_EVENT**, **RECONFIG**,
RECONFIG_PASSWORD, **RECONFIG_COMPLEX_FILTER**, **UPLOAD_WINE_DLLS**, **UPDATE_FIRMWARE**,
CHANGE_LICENCES, **PRIVILEGES_END** }
- enum **Reason** {
R_UNSUPPORTED_BIT_MASK, **R_BIT_MASK_OVERLAP**, **R_UNSUPPORTED_COMP-U_TAB**, **R_FORBIDDEN_TAB_VALUE**,
R_UNKNOWN }
- enum **BPNGLoggerStatus** {
LS_OK = 0, **LS_ERROR** = 1, **LS_NOSYNC** = 2, **LS_WARNING** = 3,
LS_FWUPDATE = 4, **LS_MEM** = 5, **LS_RING** = 6, **LS_UNDEFINED** = -1 }
Logger status.
- enum **BPNGDeviceType** {
DEV_BP2, **DEV_BPMINI**, **DEV_BP2_V1X**, **DEV_BP2_V2X**,
DEV_RC_TOUCH, **DEV_BP_REMOTE**, **DEV_BP_TOUCH**, **DEV_RAPID**,
DEV_TRACE_COLL, **DEV_TSL** = 0x80, **DEV_UNKNOWN** = 0xFF }
Enumeration of Telemotive's next generation data loggers.
- enum **DataSpanType** { **DST_IDSPAN** = 0, **DST_TIMESPAN** = 1 }
Types for [DataSpan](#).

7.1.1 Detailed Description

Defines for Telemotive Client Library.

Author

Markus van Pinxteren

Date

12.05.2010

7.1.2 Enumeration Type Documentation

7.1.2.1 enum BPNGBugreportMode

Mode for the `IBPNG::downloadBugReport()` function.

Enumerator

- BR_FULL_WO_TRACES** Full bug report without traces.
- BR_ONLY_LOGS** Only log files are downloaded.
- BR_FDB_RDB** only FDB and RDB are downloaded
- BR_ONLY_CLIENT** only client logs are stored
- BR_FULL_ALL_TRACES** Full bug report with all traces files.
- BR_FULL_TIMESPAN_TRACES** Full bugreport with trace file of a specified time span.

7.1.2.2 enum BPNGDeviceType

Enumeration of Telemotives next generation data loggers.

Enumerator

DEV_BP2 **Deprecated** For blue PiraT 2 devices use type *DEV_BP2_V1X*, for new blue PiraT 2 5E devices use *DEV_BP2_V2X*

DEV_BPMINI blue PiraT mini devices

DEV_BP2_V1X standard blue PiraT 2 device

DEV_BP2_V2X blue PiraT 2 5E device

DEV_RC_TOUCH Remote Control Touch.

DEV_BP_REMOTE blue PiraT Remote

DEV_BP_TOUCH blue PiraT Touch

DEV_RAPID blue PiraT Rapid

DEV_TRACE_COLL Trace Collector.

DEV_TSL internal use only! don't use!

7.1.2.3 enum BPNGErrCode

enum Error codes

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

Enumerator

BPNG_NOERR no error

BPNG_LOGGER_NOT_FOUND The IP address the lib wanted to connect was not found.

BPNG_NOT_CONNECTED A function call failed because the logger was not connected.

BPNG_CONNECT_FTP_FAILED Establishing the ftp connection failed.

BPNG_CONNECT_TMPBUS_FAILED Establishing the TMP (Telemotive Protocol) bus connection failed.

BPNG_TMPBUS_NOT_CONNECTED TMP bus is not connected.

BPNG_AMBIGUOUS_IP multiple devices with same IP available

BPNG_FAILED_TO_CONNECT_STREAMING Streaming feature could not be connected.

BPNG_FTP_NOT_CONNECTED FTP is not connected.

BPNG_FTP_SERVER_NOT_FOUND FTP server is not found.

BPNG_FTP_LOGIN_FAILED FTP login failed.

BPNG_FTP_REMOTE_PATH_NOT_FOUND A requested path on the FTP server is not found.

BPNG_FTP_READ_REMOTE_FILE_ERROR Can't read a file on the FTP server.

BPNG_FTP_WRITE_REMOTE_FILE_ERROR Can't write a file on the FTP server.

BPNG_FTP_TRANSFER_USER_CANCELED FTP file transfer was canceled by the user.

- BPNG_FTP_CREATE_REMOTE_DIR_ERROR** Can't create the directory on the FTP server.
- BPNG_FTP_REMOVE_REMOTE_DIR_ERROR** Can't remove the directory on the FTP server.
- BPNG_FTP_REMOVE_REMOTE_FILE_ERROR** Can't remove the file on the FTP server.
- BPNG_FTP_CHANGE_CWD_ERROR** Can't change the current working directory on the FTP server.
- BPNG_TMPBUS_COPYRDB_ERROR** Failed to copy the reference data base to the logger's tmp directory.
- BPNG_TMPBUS_SEND_MSG_ERROR** Failed to send a TMP bus request message.
- BPNG_TMPBUS_REQUEST_ERROR** The TMP bus request execution failed.
- BPNG_FAILED_TO_CREATE_LOCAL_FILE_OR_DIRECTORY** Failed to create local file or directory.
- BPNG_LOCAL_PATH_NOT_FOUND** Local path not found.
- BPNG_READ_LOCAL_FILE_ERROR** Failed to read local file.
- BPNG_WRITE_LOCAL_FILE_ERROR** Failed to write local file.
- BPNG_FILE_EXISTS_ERROR** Local file already exists.
- BPNG_DIR_EXISTS_ERROR** Local directory already exists.
- BPNG_TARGET_PATH_TOO_LONG** Specified path exceeds the max. valid length (e.g. 260 for Windows systems)
- BPNG_ZIP_EXCEEDS_FATFS_MAX** ZIP file exceeds max size for FAT32 file systems.
- BPNG_XML_PARSER_ERROR** Error while parsing xml file.
- BPNG_INITIALISATION_ERROR** BPNGClient instance is not initialised or with the wrong function. Use [IBPNGClient::initOnline](#) for data download or conversion directly from the device and [IBPNGClient::iniOffline](#) for data conversion from an offline data set.
- BPNG_RDB_SQLITE_QUERY_ERROR** Error when trying to read data from the rdb.
- BPNG_RDB_OPEN_FAILED** Failed to open the reference data base.
- BPNG_CONVERSION_ERRORS** Multiple conversion errors. Use [IBPNGClient::getNumConversionErrors\(\)](#) and [IBPNGClient::getConversionError\(\)](#) for further information
- BPNG_CONV_SET_NOT_FOUND** The passed conversion set pointer was not created with this [IBPNGClient](#) instance and dus could not be found.
- BPNG_NOTHING_TO_CONVERT** There is no data available that could be converted. Check the specified time/id spans.
- BPNG_TMT_FILE_ID_ERROR** Invalid TMT/XTMT file id while trying to convert data.
- BPNG_TMT_FORMAT_ERROR_VERSION** The TMT/XTMT version of the trace file is not supported by this lib version.
- BPNG_TMT_FORMAT_ERROR_TS** Missing FileTimeMessage in header of TMT/XTMT file.
- BPNG_INVALID_MESSAGE_ERROR** Invalid messages found in trace file(s).
- BPNG_INVALID_MESSAGE_ID** Invalid message id found in trace file(s).
- BPNG_INVALID_MESSAGE_TS** Invalid message ts found in trace file(s).
- BPNG_INVALID_MESSAGE_SUBID** Invalid message sub id found in trace file(s).
- BPNG_INVALID_MESSAGE_LEN** Invalid message length found in trace file(s).
- BPNG_CONV_FORMAT_ERROR** Invalid format assignment or mismatching recorded trace data for the specified conversion format.

- BPNG_DOWNLOAD_ERRORS** Multiple download errors. Use [IBPNGClient::getNumDownloadErrors\(\)](#) and [IBPNGClient::getDownloadError\(\)](#) for further information
- BPNG_NOTHING_TO_DOWNLOAD** There is no data available that could be downloaded. Check the specified time/id spans.
- BPNG_INVALID_OFFLINE_SET** Failed to initialise the [IBPNGClient](#) from the passed offline data set.
- BPNG_PARAMETER_MISMATCH** currently not used
- BPNG_FW_VERSION_CHECK_ERROR** The verification of the new firmware at the end of a firmware update failed.
- BPNG_USER_CANCELLED** currently not used
- BPNG_MIN_VERSION_ERROR** The current library version does not suffice the the required min version written to [BPNGError::msg](#).
- BPNG_EXCEPTION** Some kind of unhandled exception was thrown.
- BPNG_INCOMPATIBLE_RDB** The logger's or offline data set's RDB-Version is incompatible to this library version.
- BPNG_UNSPECIFIED_ERROR** An unspecified error occurred.
- 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.

7.1.2.4 enum BPNGLoggerStatus

Logger status.

Enumerator

- LS_OK** Device is ok.
- LS_ERROR** Device has at least one active error.
- LS_NOSYNC** Device is configured as slave but no master is found.
- LS_WARNING** Device has at least one active warning.
- LS_FWUPDATE** Firmware update in progress.
- LS_MEM** Internal storage of device is full. Ring buffer deactivated or full with protected trace files.
- LS_RING** Internal storage of device is full. Ring buffer is activated.

7.1.2.5 enum BPNGWarningCode

Warning codes.

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

Enumerator

BPNG_WARNING_CLOSE_TRACE_FILES no warning Failed to close the current trace files on the logger device when trying to execute [IBPNGClient::initOnline\(\)](#)

BPNG_WARNING_MESSAGES_NOT_CONVERTED In case of protocol mismatch between recorded data and target format or unsupported message sub types, it is possible that some messages can not be converted to the selected format.

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

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

BPNG_WARNING_RECOVERING_FAILED Recovering trace files from a previous power down failed.

7.1.2.6 enum ChannelType

Currently supported interfaces.

Enumerator

CH_UNDEFINED undefined channel type

OBSOLETE_CH_CANLS CAN low speed interface.

CH_CAN CAN high speed interface.

CH_LIN LIN interface.

CH_SERIAL Serial interface.

CH_ETHERNET Ethernet interface.

CH_FLEXRAY Flexray interface.

CH_MOST25_CTRL MOST 25 control channel.

CH_MOST25_MDP MOST 25 data packet channel (MDP)

CH_MOST25_SYNC MOST 25 synchronous channel (streaming data)

CH_MOST150_CTRL MOST 150 control channel.

CH_MOST150_MDP MOST 150 data packet channel (MDP)

CH_MOST150_MEP MOST 150 ethernet packet channel (MEP)

CH_MOST150_STREAM MOST 150 synchronous channel (streaming data)

CH_ANALOG_IN Analog in.

CH_DIGITAL_IN Digital in.

CH_CAMERA Camera channel.

CH_CCPXCP CCP XCP.

CH_DIAG Diagnose, currently not used.

CH_GPS Global Positioning System.

CH_ECL Electronic Control Line.

CH_TTY TTY channel for QXDM.

7.1.2.7 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

7.1.2.8 enum Reason

Enumerator

R_UNSUPPORTED_BIT_MASK DBC file don't support bit operations with a bit mask.
R_BIT_MASK_OVERLAP Bit mask is incorrect and cause a overlap with at least one other signal.
R_UNSUPPORTED_COMPU_TAB DBC file don't support all compu tab types; only tab will ignored, not the signal itself!
R_FORBIDDEN_TAB_VALUE DBC file don't support all possible values of a compu tab; only tab will ignored, not the signal itself!
R_UNKNOWN Unknown reason.

7.2 BPNGLoggerDetector.hh File Reference

Logger Detector Sample.

```
#include "IBPNGClient.h"  
#include "IBPNGClientListener.h"  
#include <vector>  
#include <string>  
#include <iostream>  
#include <sstream>
```

Classes

- class [TSLCluster](#)
- class [BPNGLoggerDetector](#)

7.2.1 Detailed Description

Logger Detector Sample.

7.3 IBPNGClient.h File Reference

Interface class for the BPNGClient DLL.

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

Classes

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

Functions

- DECLDIR const char *WINAPI [getLibVersion](#) ()
Returns the current client library version.
- DECLDIR [IBPNGClient](#) *WINAPI [getBPNGClient](#) (const char *name="")
Factory function that creates instances of BPNGClient giving away ownership.
- DECLDIR [IBPNGClient](#) *WINAPI [getTSLClient](#) (int numTSLMember)
Factory function returning an [IBPNGClient](#) instance for working with a TSL logger cluster.
- DECLDIR [BPNGErrorCode](#) WINAPI [getNumTSLMemberFromOfflineDataSet](#) (const char *offline-Path, 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 numBackup-Files)
- 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.

7.3.1 Detailed Description

Interface class for the BPNGClient DLL.

Author

Markus van Pinxteren

Date

21.04.2010

7.3.2 Function Documentation

7.3.2.1 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](#)

7.3.2.2 DECLDIR IClientProperties* WINAPI createNewClientProperties ()

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

See Also

[IClientProperties](#), setClientProperties()

7.3.2.3 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.

7.3.2.4 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.

7.3.2.5 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\(\)](#)

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

Read out the number of TSL members from a offline data set. Needed for [getTSLClient\(int num-TSLMember\)](#).

7.3.2.7 DECLDIR IBPNGClient* WINAPI getTSLClient (int *numTSLMember*)

Factory function returning an [IBPNGClient](#) instance for working with a TSL logger cluster.

The instance is created on the heap and the allocated memory must be freed by the calling application. You must pass the number of devices in TSL as argument. For online operations you can use the number of devices in a TSL chain from logger detection. On offline operations you can use the function [getNumTSLMemberFromOfflineDataSet\(const char* offlinePath, int* numMember\)](#) to get the number of TSL members.

See Also

[BPNGLoggerDetector](#)
[TSLCluster](#)
[IBPNGClient::release\(\)](#)

7.3.2.8 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.

7.3.2.9 DECLDIR void WINAPI writeLogToCout (bool *flag*)

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

7.4 IBPNGClientListener.h File Reference

Interface class for the BPNGClient listener.


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

Classes

- struct [IBPNGClientListener](#)

7.4.1 Detailed Description

Interface class for the BPNGClient listener.

Author

Markus van Pinxteren

Date

12.05.2010

7.5 IClientProperties.h File Reference

Interface for client properties.

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

Classes

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

7.5.1 Detailed Description

Interface for client properties.

Author

Markus van Pinxteren

Date

20.03.2014

7.6 RdbDefines.h File Reference

Public interfaces for Telemotive Reference Database access.

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

Classes

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

Enumerations

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

7.6.1 Detailed Description

Public interfaces for Telemotive Reference Database access.

7.6.2 Enumeration Type Documentation

7.6.2.1 enum RdbEventType

See Also

tmlib's eventID.hh

Enumerator

STARTUP bp2 startup
SHUTDOWN bp2 shutdown
MARKER Marker set.
INFO Info is set.

SLAVE_OFFSET cascading slave offset
SLAVE_TO_MASTER cascading slave to master
DATA_DELETED All data and data space is deleted.
TIME_SET bp2 time was set
NEW_TIME new time
SUDDEN_DEATH no "real" shutdown was found after startup.
TSL_SLAVE_OFFSET slave is synced with master.
TSL_SLAVE_TO_MASTER slave is not synced with master.
TSL_SESSION_START start of a tsl synchronized session
TSL_SESSION_END end of a tsl synchronized session
CONFIG configuration has been updated
WAKEUP bpng wake-up source
START_TESTDRIVE start an easy track test drive
STOP_TESTDRIVE stop an easy track test drive
TESTDRIVE_INFO misc. info event for test drive data like testname, vin, map-version, reproducibility, etc.

7.7 RdbEventList.hh File Reference

[IRdbEvent](#) wrapper.

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

Classes

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

7.7.1 Detailed Description

[IRdbEvent](#) wrapper.

Index

- activateGatewayLoggerDetection
 - IBPNGClient, [32](#)
- addAnalogPortSettings
 - IClientProperties, [60](#)
- addChannel
 - IConversionSet, [63](#)
- addDevice
 - TSLCluster, [76](#)
- addLogListener
 - IBPNGClient.h, [89](#)
- addRdbldRange
 - IConversionSet, [63](#)
- addTimeSpan
 - IConversionSet, [63](#)
- assignDBCFile
 - IBPNGClient, [32](#)
- BPNG_AMBIGUOUS_IP
 - BPNGDefines.h, [83](#)
- BPNG_CONNECT_FTP_FAILED
 - BPNGDefines.h, [83](#)
- BPNG_CONNECT_TMPBUS_FAILED
 - BPNGDefines.h, [83](#)
- BPNG_CONV_FORMAT_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_CONV_SET_NOT_FOUND
 - BPNGDefines.h, [84](#)
- BPNG_CONVERSION_ERRORS
 - BPNGDefines.h, [84](#)
- BPNG_DIR_EXISTS_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_DOWNLOAD_ERRORS
 - BPNGDefines.h, [84](#)
- BPNG_ENGLISH
 - BPNGDefines.h, [87](#)
- BPNG_EXCEPTION
 - BPNGDefines.h, [85](#)
- BPNG_FAILED_TO_CONNECT_STREAMING
 - BPNGDefines.h, [83](#)
- BPNG_FAILED_TO_CREATE_LOCAL_FILE_OR_DIRECTORY
 - BPNGDefines.h, [84](#)
- BPNG_FILE_EXISTS_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_CHANGE_CWD_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_CREATE_REMOTE_DIR_ERROR
 - BPNGDefines.h, [83](#)
- BPNG_FTP_LOGIN_FAILED
 - BPNGDefines.h, [83](#)
- BPNG_FTP_NOT_CONNECTED
 - BPNGDefines.h, [83](#)
- BPNG_FTP_READ_REMOTE_FILE_ERROR
 - BPNGDefines.h, [83](#)
- BPNG_FTP_REMOTE_PATH_NOT_FOUND
 - BPNGDefines.h, [83](#)
- BPNG_FTP_REMOVE_REMOTE_DIR_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_REMOVE_REMOTE_FILE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_FTP_SERVER_NOT_FOUND
 - BPNGDefines.h, [83](#)
- BPNG_FTP_TRANSFER_USER_CANCELED
 - BPNGDefines.h, [83](#)
- BPNG_FTP_WRITE_REMOTE_FILE_ERROR
 - BPNGDefines.h, [83](#)
- BPNG_FW_VERSION_CHECK_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_GERMAN
 - BPNGDefines.h, [87](#)
- BPNG_INCOMPATIBLE_RDB
 - BPNGDefines.h, [85](#)
- BPNG_INCONSISTENT_TSL_FWVERSIONS
 - BPNGDefines.h, [85](#)
- BPNG_INITIALISATION_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_INVALID_MESSAGE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_INVALID_MESSAGE_ID
 - BPNGDefines.h, [84](#)
- BPNG_INVALID_MESSAGE_LEN
 - BPNGDefines.h, [84](#)
- BPNG_INVALID_MESSAGE_SUBID
 - BPNGDefines.h, [84](#)
- BPNG_INVALID_MESSAGE_TS
 - BPNGDefines.h, [84](#)

- BPNG_INVALID_OFFLINE_SET
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_RPC_COMMAND
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_TSL_CASCDING
 - BPNGDefines.h, [85](#)
- BPNG_INVALID_TSL_CLUSTER
 - BPNGDefines.h, [85](#)
- BPNG_LOCAL_PATH_NOT_FOUND
 - BPNGDefines.h, [84](#)
- BPNG_LOGGER_NOT_FOUND
 - BPNGDefines.h, [83](#)
- BPNG_MIN_VERSION_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_NOERR
 - BPNGDefines.h, [83](#)
- BPNG_NOT_CONNECTED
 - BPNGDefines.h, [83](#)
- BPNG_NOTHING_TO_CONVERT
 - BPNGDefines.h, [84](#)
- BPNG_NOTHING_TO_DOWNLOAD
 - BPNGDefines.h, [85](#)
- BPNG_NOTHING_TO_TEST_REPORT
 - BPNGDefines.h, [85](#)
- BPNG_PARAMETER_MISMATCH
 - BPNGDefines.h, [85](#)
- BPNG_RDB_OPEN_FAILED
 - BPNGDefines.h, [84](#)
- BPNG_RDB_SQLITE_QUERY_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_READ_LOCAL_FILE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_TARGET_PATH_TOO_LONG
 - BPNGDefines.h, [84](#)
- BPNG_TMPBUS_COPYRDB_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_TMPBUS_NOT_CONNECTED
 - BPNGDefines.h, [83](#)
- BPNG_TMPBUS_REQUEST_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_TMPBUS_SEND_MSG_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_TMT_FILE_ID_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_TMT_FORMAT_ERROR_TS
 - BPNGDefines.h, [84](#)
- BPNG_TMT_FORMAT_ERROR_VERSION
 - BPNGDefines.h, [84](#)
- BPNG_UNSPECIFIED_ERROR
 - BPNGDefines.h, [85](#)
- BPNG_USER_CANCELLED
 - BPNGDefines.h, [85](#)
- BPNG_WARNING_CLOSE_TRACE_FILES
 - BPNGDefines.h, [86](#)
- BPNG_WARNING_MESSAGES_NOT_CONVERTED
 - BPNGDefines.h, [86](#)
- BPNG_WARNING_NO_ESO_TRACE
 - BPNGDefines.h, [86](#)
- BPNG_WARNING_RECOVERING_FAILED
 - BPNGDefines.h, [86](#)
- BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONES
 - BPNGDefines.h, [86](#)
- BPNG_WRITE_LOCAL_FILE_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_XML_PARSER_ERROR
 - BPNGDefines.h, [84](#)
- BPNG_ZIP_EXCEEDS_FATFS_MAX
 - BPNGDefines.h, [84](#)
- BPNGDefines.h
 - BPNG_AMBIGUOUS_IP, [83](#)
 - BPNG_CONNECT_FTP_FAILED, [83](#)
 - BPNG_CONNECT_TMPBUS_FAILED, [83](#)
 - BPNG_CONV_FORMAT_ERROR, [84](#)
 - BPNG_CONV_SET_NOT_FOUND, [84](#)
 - BPNG_CONVERSION_ERRORS, [84](#)
 - BPNG_DIR_EXISTS_ERROR, [84](#)
 - BPNG_DOWNLOAD_ERRORS, [84](#)
 - BPNG_ENGLISH, [87](#)
 - BPNG_EXCEPTION, [85](#)
 - BPNG_FAILED_TO_CONNECT_STREAMING, [83](#)
 - BPNG_FAILED_TO_CREATE_LOCAL_FILE_OR_DIRECTORY, [84](#)
 - BPNG_FILE_EXISTS_ERROR, [84](#)
 - BPNG_FTP_CHANGE_CWD_ERROR, [84](#)
 - BPNG_FTP_CREATE_REMOTE_DIR_ERROR, [83](#)
 - BPNG_FTP_LOGIN_FAILED, [83](#)
 - BPNG_FTP_NOT_CONNECTED, [83](#)
 - BPNG_FTP_READ_REMOTE_FILE_ERROR, [83](#)
 - BPNG_FTP_REMOTE_PATH_NOT_FOUND, [83](#)
 - BPNG_FTP_REMOVE_REMOTE_DIR_ERROR, [84](#)
 - BPNG_FTP_REMOVE_REMOTE_FILE_ERROR, [84](#)
 - BPNG_FTP_SERVER_NOT_FOUND, [83](#)
 - BPNG_FTP_TRANSFER_USER_CANCELLED, [83](#)
 - BPNG_FTP_WRITE_REMOTE_FILE_ERROR, [83](#)
 - BPNG_FW_VERSION_CHECK_ERROR, [85](#)
 - BPNG_GERMAN, [87](#)
 - BPNG_INCOMPATIBLE_RDB, [85](#)

- BPNG_INCONSISTENT_TSL_FWVERSIONS, 85
- BPNG_INITIALISATION_ERROR, 84
- BPNG_INVALID_MESSAGE_ERROR, 84
- BPNG_INVALID_MESSAGE_ID, 84
- BPNG_INVALID_MESSAGE_LEN, 84
- BPNG_INVALID_MESSAGE_SUBID, 84
- BPNG_INVALID_MESSAGE_TS, 84
- BPNG_INVALID_OFFLINE_SET, 85
- BPNG_INVALID_RPC_COMMAND, 85
- BPNG_INVALID_TSL_CASCDING, 85
- BPNG_INVALID_TSL_CLUSTER, 85
- BPNG_LOCAL_PATH_NOT_FOUND, 84
- BPNG_LOGGER_NOT_FOUND, 83
- BPNG_MIN_VERSION_ERROR, 85
- BPNG_NOERR, 83
- BPNG_NOT_CONNECTED, 83
- BPNG_NOTHING_TO_CONVERT, 84
- BPNG_NOTHING_TO_DOWNLOAD, 85
- BPNG_NOTHING_TO_TEST_REPORT, 85
- BPNG_PARAMETER_MISMATCH, 85
- BPNG_RDB_OPEN_FAILED, 84
- BPNG_RDB_SQLITE_QUERY_ERROR, 84
- BPNG_READ_LOCAL_FILE_ERROR, 84
- BPNG_TARGET_PATH_TOO_LONG, 84
- BPNG_TMPBUS_COPYRDB_ERROR, 84
- BPNG_TMPBUS_NOT_CONNECTED, 83
- BPNG_TMPBUS_REQUEST_ERROR, 84
- BPNG_TMPBUS_SEND_MSG_ERROR, 84
- BPNG_TMT_FILE_ID_ERROR, 84
- BPNG_TMT_FORMAT_ERROR_TS, 84
- BPNG_TMT_FORMAT_ERROR_VERSION, 84
- BPNG_UNSPECIFIED_ERROR, 85
- BPNG_USER_CANCELLED, 85
- BPNG_WARNING_CLOSE_TRACE_FILES, 86
- BPNG_WARNING_MESSAGES_NOT_CONVERTED, 86
- BPNG_WARNING_NO_ESO_TRACE, 86
- BPNG_WARNING_RECOVERING_FAILED, 86
- BPNG_WARNING_TSL_WITH_DIFFERENT_TIMEZONES, 86
- BPNG_WRITE_LOCAL_FILE_ERROR, 84
- BPNG_XML_PARSER_ERROR, 84
- BPNG_ZIP_EXCEEDS_FATFS_MAX, 84
- BR_FDB_RDB, 82
- BR_FULL_ALL_TRACES, 82
- BR_FULL_TIMESPAN_TRACES, 82
- BR_FULL_WO_TRACES, 82
- BR_ONLY_CLIENT, 82
- BR_ONLY_LOGS, 82
- CH_ANALOG_IN, 86
- CH_CAMERA, 86
- CH_CAN, 86
- CH_CCPXCP, 86
- CH_DIAG, 86
- CH_DIGITAL_IN, 86
- CH_ECL, 86
- CH_ETHERNET, 86
- CH_FLEXRAY, 86
- CH_GPS, 86
- CH_LIN, 86
- CH_MOST150_CTRL, 86
- CH_MOST150_MDP, 86
- CH_MOST150_MEP, 86
- CH_MOST150_STREAM, 86
- CH_MOST25_CTRL, 86
- CH_MOST25_MDP, 86
- CH_MOST25_SYNC, 86
- CH_SERIAL, 86
- CH_TTY, 86
- CH_UNDEFINED, 86
- DEV_BP2, 83
- DEV_BP2_V1X, 83
- DEV_BP2_V2X, 83
- DEV_BP_REMOTE, 83
- DEV_BP_TOUCH, 83
- DEV_BPMINI, 83
- DEV_RAPID, 83
- DEV_RC_TOUCH, 83
- DEV_TRACE_COLL, 83
- DEV_TSL, 83
- LS_ERROR, 85
- LS_FWUPDATE, 85
- LS_MEM, 85
- LS_NOSYNC, 85
- LS_OK, 85
- LS_RING, 85
- LS_WARNING, 85
- OBSOLETE_CH_CANLS, 86
- R_BIT_MASK_OVERLAP, 87
- R_FORBIDDEN_TAB_VALUE, 87
- R_UNKNOWN, 87
- R_UNSUPPORTED_BIT_MASK, 87
- R_UNSUPPORTED_COMPU_TAB, 87
- BR_FDB_RDB
 - BPNGDefines.h, 82
- BR_FULL_ALL_TRACES
 - BPNGDefines.h, 82
- BR_FULL_TIMESPAN_TRACES
 - BPNGDefines.h, 82
- BR_FULL_WO_TRACES
 - BPNGDefines.h, 82
- BR_ONLY_CLIENT
 - BPNGDefines.h, 82
- BR_ONLY_LOGS

- BPNGDefines.h, 82
- BUGREPORT
 - TSLCluster, 75
- BPNGBugreportMode
 - BPNGDefines.h, 82
- BPNGDefines.h, 78
 - BPNGBugreportMode, 82
 - BPNGDeviceType, 82
 - BPNGErrCode, 83
 - BPNGLoggerStatus, 85
 - BPNGWarningCode, 85
 - ChannelType, 86
 - LanguageID, 86
 - Reason, 87
- BPNGDeviceType
 - BPNGDefines.h, 82
- BPNGErrCode
 - BPNGDefines.h, 83
- BPNGError, 21
- BPNGLoggerDetector, 21
 - BPNGLoggerDetector, 23
 - BPNGLoggerDetector, 23
 - getLoggerList, 23
 - getOverwritingPermission, 23
 - getTSLs, 23
 - onBPNGDeviceDetected, 24
 - onBPNGDeviceDisappeared, 24
 - onBPNGDeviceStateChange, 24
 - onConversionStart, 24
 - onCriticalDiskSpace, 24
 - onDataRecoverProgress, 25
 - onDownloadStart, 25
 - onExtractionPasswordRequired, 25
 - onGetLogReportProgress, 26
 - onInvalidPwConfigFound, 26
 - onLogInDataRequired, 26
 - onProgressConversion, 26
 - onProgressDataDownload, 27
 - onStatusMessage, 27
 - onTargetPathTooLong, 27
 - onWarning, 27
- BPNGLoggerDetector.hh, 87
- BPNGLoggerStatus
 - BPNGDefines.h, 85
- BPNGWarningCode
 - BPNGDefines.h, 85
- begin
 - TSLCluster, 76
- CH_ANALOG_IN
 - BPNGDefines.h, 86
- CH_CAMERA
 - BPNGDefines.h, 86
- CH_CAN
 - BPNGDefines.h, 86
- CH_CCPXCP
 - BPNGDefines.h, 86
- CH_DIAG
 - BPNGDefines.h, 86
- CH_DIGITAL_IN
 - BPNGDefines.h, 86
- CH_ECL
 - BPNGDefines.h, 86
- CH_ETHERNET
 - BPNGDefines.h, 86
- CH_FLEXRAY
 - BPNGDefines.h, 86
- CH_GPS
 - BPNGDefines.h, 86
- CH_LIN
 - BPNGDefines.h, 86
- CH_MOST150_CTRL
 - BPNGDefines.h, 86
- CH_MOST150_MDP
 - BPNGDefines.h, 86
- CH_MOST150_MEP
 - BPNGDefines.h, 86
- CH_MOST150_STREAM
 - BPNGDefines.h, 86
- CH_MOST25_CTRL
 - BPNGDefines.h, 86
- CH_MOST25_MDP
 - BPNGDefines.h, 86
- CH_MOST25_SYNC
 - BPNGDefines.h, 86
- CH_SERIAL
 - BPNGDefines.h, 86
- CH_TTY
 - BPNGDefines.h, 86
- CH_UNDEFINED
 - BPNGDefines.h, 86
- CONFIG
 - RdbDefines.h, 93
 - TSLCluster, 75
- CONVERSION
 - TSLCluster, 75
- ChannelType
 - BPNGDefines.h, 86
- connectLogger
 - IBPNGClient, 32, 33
- ConnectionType
 - TSLCluster, 75
- convertData
 - IBPNGClient, 33
- createNewClientProperties
 - IBPNGClient.h, 89
- createNewConversionSet
 - IBPNGClient, 34

- createTestReport
 - IBPNGClient, [34](#)
- DATA_DELETED
 - RdbDefines.h, [93](#)
- DEV_BP2
 - BPNGDefines.h, [83](#)
- DEV_BP2_V1X
 - BPNGDefines.h, [83](#)
- DEV_BP2_V2X
 - BPNGDefines.h, [83](#)
- DEV_BP_REMOTE
 - BPNGDefines.h, [83](#)
- DEV_BP_TOUCH
 - BPNGDefines.h, [83](#)
- DEV_BPMINI
 - BPNGDefines.h, [83](#)
- DEV_RAPID
 - BPNGDefines.h, [83](#)
- DEV_RC_TOUCH
 - BPNGDefines.h, [83](#)
- DEV_TRACE_COLL
 - BPNGDefines.h, [83](#)
- DEV_TSL
 - BPNGDefines.h, [83](#)
- DOWNLOAD
 - TSLCluster, [75](#)
- DataSpan, [28](#)
- deleteAllData
 - IBPNGClient, [34](#)
- deleteData
 - IBPNGClient, [35](#)
- deviceType
 - OnlineLoggerInfo, [72](#)
- downloadBugReport
 - IBPNGClient, [35](#)
- downloadDataSpans
 - IBPNGClient, [36](#)
- end
 - TSLCluster, [76](#)
- FW_UPDATE
 - TSLCluster, [75](#)
- filterSignals
 - IBPNGClient, [36](#)
- filterSignalsFromOfflineData
 - IBPNGClient, [36](#)
- flashDeviceLED
 - IBPNGClient, [37](#)
- freeClientPropertiesMemory
 - IBPNGClient.h, [89](#)
- freeConversionSetMemory
 - IBPNGClient.h, [89](#)
- getAvailableFormats
 - IBPNGClient, [37](#)
- getBPNGClient
 - IBPNGClient.h, [89](#)
- getClientProperties
 - IBPNGClient, [37](#)
- getComment
 - IRdbEvent, [67](#)
- getConfig
 - IBPNGClient, [37](#)
- getConfigPath
 - IBPNGClient, [38](#)
- getConnectionVector
 - TSLCluster, [76](#)
- getConversionError
 - IBPNGClient, [38](#)
- getDeviceName
 - IBPNGClient, [38](#)
- getDownloadError
 - IBPNGClient, [38](#)
- getEventList
 - IBPNGClient, [38](#)
- getFalseMeasureSignals
 - IBPNGClient, [39](#)
- getLastError
 - IBPNGClient, [39](#)
- getLicenses
 - IBPNGClient, [39](#)
- getLoggerChannels
 - IBPNGClient, [40](#)
- getLoggerList
 - BPNGLoggerDetector, [23](#)
- getMemoryFillLevel
 - IBPNGClient, [40](#)
- getNumConversionErrors
 - IBPNGClient, [40](#)
- getNumDownloadErrors
 - IBPNGClient, [41](#)
- getNumMembers
 - TSLCluster, [76](#)
- getNumTSLMemberFromOfflineDataSet
 - IBPNGClient.h, [90](#)
- getOverwritingPermission
 - BPNGLoggerDetector, [23](#)
 - IBPNGClientListener, [51](#)
- getPwdFile
 - IBPNGClient, [41](#)
- getReferenceDataBasePath
 - IBPNGClient, [41](#)
- getTSLClient
 - IBPNGClient.h, [90](#)
- getTSLName
 - TSLCluster, [77](#)
- getTSLs

- BPNGLoggerDetector, 23
- getTimeZone
 - IRdbEvent, 67
- getTraceBlockList
 - IBPNGClient, 41
- getUniqueld
 - IRdbEvent, 67
- getVersions
 - IBPNGClient, 42
- INFO
 - RdbDefines.h, 92
- IBPNGClient, 28
 - activateGatewayLoggerDetection, 32
 - assignDBCFile, 32
 - connectLogger, 32, 33
 - convertData, 33
 - createNewConversionSet, 34
 - createTestReport, 34
 - deleteAllData, 34
 - deleteData, 35
 - downloadBugReport, 35
 - downloadDataSpans, 36
 - filterSignals, 36
 - filterSignalsFromOfflineData, 36
 - flashDeviceLED, 37
 - getAvailableFormats, 37
 - getClientProperties, 37
 - getConfig, 37
 - getConfigPath, 38
 - getConversionError, 38
 - getDeviceName, 38
 - getDownloadError, 38
 - getEventList, 38
 - getFalseMeasureSignals, 39
 - getLastError, 39
 - getLicenses, 39
 - getLoggerChannels, 40
 - getMemoryFillLevel, 40
 - getNumConversionErrors, 40
 - getNumDownloadErrors, 41
 - getPwdFile, 41
 - getReferenceDataBasePath, 41
 - getTraceBlockList, 41
 - getVersions, 42
 - initOffline, 42
 - initOnline, 43
 - isPasswordProtectionSupported, 43
 - keepLoggerAlive, 43
 - reconfigLogger, 44
 - release, 45
 - removeAllLicenses, 45
 - restartDevice, 45
 - scanNetworkForLogger, 46
 - setClientProperties, 46
 - setDefaultConfig, 46
 - setInfoEvent, 46
 - setMarker, 47
 - setPwdFile, 47
 - setTime, 47
 - shutdownDevice, 47
 - synchronizeRdb, 47
 - updateFirmware, 48
 - updateLicenses, 49
- IBPNGClient.h, 88
 - addLogListener, 89
 - createNewClientProperties, 89
 - freeClientPropertiesMemory, 89
 - freeConversionSetMemory, 89
 - getBPNGClient, 89
 - getNumTSLMemberFromOfflineDataSet, 90
 - getTSLClient, 90
 - writeLogFile, 90
 - writeLogToCout, 90
- IBPNGClientListener, 50
 - getOverwritingPermission, 51
 - onBPNGDeviceDetected, 51
 - onBPNGDeviceDisappeared, 51
 - onBPNGDeviceStateChange, 51
 - onConversionStart, 51
 - onCriticalDiskSpace, 52
 - onDataRecoverProgress, 52
 - onDownloadStart, 52
 - onExtractionPasswordRequired, 53
 - onGetLogReportProgress, 53
 - onInvalidPwConfigFound, 53
 - onLogInDataRequired, 53
 - onProgressConversion, 54
 - onProgressDataDownload, 54
 - onStatusMessage, 54
 - onTargetPathTooLong, 55
 - onWarning, 55
- IBPNGClientListener.h, 90
- IChannel, 55
- IChannelList, 56
- IClientProperties, 56
 - addAnalogPortSettings, 60
 - setCANPseudoMsgTimeStampProperties, 60
 - setCANPseudoMsgTriggerProperties, 60
 - setCommonProperties, 61
 - setMOSTPseudoMsgProperties, 61
- IClientProperties.h, 91
- IConversionSet, 62
 - addChannel, 63
 - addRdbldRange, 63
 - addTimeSpan, 63
- IFalseMeasureSignal, 64
- IFalseMeasureSignalList, 64

- IFormatInfo, [65](#)
- IFormatList, [66](#)
- IRdbEvent, [66](#)
 - getComment, [67](#)
 - getTimeZone, [67](#)
 - getUniqueId, [67](#)
- IRdbEventList, [67](#)
- IRdbTraceBlock, [67](#)
- IRdbTraceBlockList, [68](#)
- ITesttoolsChannel, [68](#)
- ITesttoolsChannelList, [69](#)
- initOffline
 - IBPNGClient, [42](#)
- initOnline
 - IBPNGClient, [43](#)
- isPasswordProtectionSupported
 - IBPNGClient, [43](#)
- keepLoggerAlive
 - IBPNGClient, [43](#)
- LS_ERROR
 - BPNGDefines.h, [85](#)
- LS_FWUPDATE
 - BPNGDefines.h, [85](#)
- LS_MEM
 - BPNGDefines.h, [85](#)
- LS_NOSYNC
 - BPNGDefines.h, [85](#)
- LS_OK
 - BPNGDefines.h, [85](#)
- LS_RING
 - BPNGDefines.h, [85](#)
- LS_WARNING
 - BPNGDefines.h, [85](#)
- LanguageID
 - BPNGDefines.h, [86](#)
- LogInData, [70](#)
- loggerStatus
 - OnlineLoggerInfo, [72](#)
- MARKER
 - RdbDefines.h, [92](#)
- MemoryFillLevel, [70](#)
- NEW_TIME
 - RdbDefines.h, [93](#)
- OBSOLETE_CH_CANLS
 - BPNGDefines.h, [86](#)
- onBPNGDeviceDetected
 - BPNGLoggerDetector, [24](#)
 - IBPNGClientListener, [51](#)
- onBPNGDeviceDisappeared
 - BPNGLoggerDetector, [24](#)
- IBPNGClientListener, [51](#)
- onBPNGDeviceStateChange
 - BPNGLoggerDetector, [24](#)
 - IBPNGClientListener, [51](#)
- onConversionStart
 - BPNGLoggerDetector, [24](#)
 - IBPNGClientListener, [51](#)
- onCriticalDiskSpace
 - BPNGLoggerDetector, [24](#)
 - IBPNGClientListener, [52](#)
- onDataRecoverProgress
 - BPNGLoggerDetector, [25](#)
 - IBPNGClientListener, [52](#)
- onDownloadStart
 - BPNGLoggerDetector, [25](#)
 - IBPNGClientListener, [52](#)
- onExtractionPasswordRequired
 - BPNGLoggerDetector, [25](#)
 - IBPNGClientListener, [53](#)
- onGetLogReportProgress
 - BPNGLoggerDetector, [26](#)
 - IBPNGClientListener, [53](#)
- onInvalidPwConfigFound
 - BPNGLoggerDetector, [26](#)
 - IBPNGClientListener, [53](#)
- onLogInDataRequired
 - BPNGLoggerDetector, [26](#)
 - IBPNGClientListener, [53](#)
- onProgressConversion
 - BPNGLoggerDetector, [26](#)
 - IBPNGClientListener, [54](#)
- onProgressDataDownload
 - BPNGLoggerDetector, [27](#)
 - IBPNGClientListener, [54](#)
- onStatusMessage
 - BPNGLoggerDetector, [27](#)
 - IBPNGClientListener, [54](#)
- onTargetPathTooLong
 - BPNGLoggerDetector, [27](#)
 - IBPNGClientListener, [55](#)
- onWarning
 - BPNGLoggerDetector, [27](#)
 - IBPNGClientListener, [55](#)
- OnlineLoggerInfo, [71](#)
 - deviceType, [72](#)
 - loggerStatus, [72](#)
- OnlineLoggerInfoStringPair, [72](#)
- print
 - TSLCluster, [77](#)
- R_BIT_MASK_OVERLAP
 - BPNGDefines.h, [87](#)
- R_FORBIDDEN_TAB_VALUE

- BPNGDefines.h, 87
- R_UNKNOWN
 - BPNGDefines.h, 87
- R_UNSUPPORTED_BIT_MASK
 - BPNGDefines.h, 87
- R_UNSUPPORTED_COMPU_TAB
 - BPNGDefines.h, 87
- RdbDefines.h
 - CONFIG, 93
 - DATA_DELETED, 93
 - INFO, 92
 - MARKER, 92
 - NEW_TIME, 93
 - SHUTDOWN, 92
 - SLAVE_OFFSET, 92
 - SLAVE_TO_MASTER, 93
 - START_TESTDRIVE, 93
 - STARTUP, 92
 - STOP_TESTDRIVE, 93
 - SUDDEN_DEATH, 93
 - TESTDRIVE_INFO, 93
 - TIME_SET, 93
 - TSL_SESSION_END, 93
 - TSL_SESSION_START, 93
 - TSL_SLAVE_OFFSET, 93
 - TSL_SLAVE_TO_MASTER, 93
 - WAKEUP, 93
- RdbDefines.h, 92
 - RdbEventType, 92
- RdbEvent2, 73
- RdbEventList, 74
- RdbEventList.hh, 93
- RdbEventType
 - RdbDefines.h, 92
- Reason
 - BPNGDefines.h, 87
- reconfigLogger
 - IBPNGClient, 44
- release
 - IBPNGClient, 45
- removeAllLicenses
 - IBPNGClient, 45
- restartDevice
 - IBPNGClient, 45
- SHUTDOWN
 - RdbDefines.h, 92
- SLAVE_OFFSET
 - RdbDefines.h, 92
- SLAVE_TO_MASTER
 - RdbDefines.h, 93
- START_TESTDRIVE
 - RdbDefines.h, 93
- STARTUP
 - RdbDefines.h, 92
- STOP_TESTDRIVE
 - RdbDefines.h, 93
- SUDDEN_DEATH
 - RdbDefines.h, 93
- scanNetworkForLogger
 - IBPNGClient, 46
- setCANPseudoMsgTimeStampProperties
 - IClientProperties, 60
- setCANPseudoMsgTriggerProperties
 - IClientProperties, 60
- setClientProperties
 - IBPNGClient, 46
- setCommonProperties
 - IClientProperties, 61
- setDefaultConfig
 - IBPNGClient, 46
- setInfoEvent
 - IBPNGClient, 46
- setMOSTPseudoMsgProperties
 - IClientProperties, 61
- setMarker
 - IBPNGClient, 47
- setPwdFile
 - IBPNGClient, 47
- setTime
 - IBPNGClient, 47
- shutdownDevice
 - IBPNGClient, 47
- synchronizeRdb
 - IBPNGClient, 47
- TESTDRIVE_INFO
 - RdbDefines.h, 93
- TIME_SET
 - RdbDefines.h, 93
- TSL_SESSION_END
 - RdbDefines.h, 93
- TSL_SESSION_START
 - RdbDefines.h, 93
- TSL_SLAVE_OFFSET
 - RdbDefines.h, 93
- TSL_SLAVE_TO_MASTER
 - RdbDefines.h, 93
- TSLCluster
 - BUGREPORT, 75
 - CONFIG, 75
 - CONVERSION, 75
 - DOWNLOAD, 75
 - FW_UPDATE, 75
- TSLCluster, 74
 - addDevice, 76
 - begin, 76
 - ConnectionType, 75

- end, [76](#)
- getConnectionVector, [76](#)
- getNumMembers, [76](#)
- getTSLName, [77](#)
- print, [77](#)
- TSLCluster, [75](#)
- TSLCluster, [75](#)
- updateFirmware
 - IBPNGClient, [48](#)
- updateLicenses
 - IBPNGClient, [49](#)
- WAKEUP
 - RdbDefines.h, [93](#)
- writeLogFile
 - IBPNGClient.h, [90](#)
- writeLogToCout
 - IBPNGClient.h, [90](#)