Telemotive AG

blue PiraT 2 Client Library 1.7.2

User's manual

Generated by Doxygen 1.8.0

Thu Sep 20 2012 10:31:26

Inhaltsverzeichnis

1 User’s manual - blue PiraT 2 Client Library 1.7.2 **1**

1.1 General 1

1.2 Functionality 1

1.3 Compiler/Linker 2

1.4 Thread safety 2

1.5 Demo project 2

Kapitel 1

User's manual - blue PiraT 2 Client Library 1.7.2

1.1 General

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

1.2 Functionality

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

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

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

1.2.1 Error handling and listener mechanism

All errors are processed by the functions' return values. If the return value states an error a call to getLastError() provides details about the error(s) occurred. Warnings are not intended to abort a process. That's why they are reported via the function IBPNGClientListener::onWarning(). It's up to the user to handle them or not.

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

1.3 Compiler/Linker

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

1.4 Thread safety

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

1.5 Demo project

The "sample" directory contains a demo project for the blue PiraT 2 Client library.
Beispiel zur Verwendung der Library:

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

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

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

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

using namespace std;

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


void main()
{
        // Get list of all currently available blue PiraT 2 devices
        BPNGLoggerDetector detector;
        vector<BP2Device> devices = detector.getLoggerList(0);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        // connect logger
        BOOL ret = client->connectLogger(device.ipAddress.c_str());
        if (ret == 0)
        {
                BPNGError err = client->getLastError();
                cout << "Failed to connect logger." << endl;
                cout << "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 == EVID_MARKER)
                {
                        markerTimeStamp = eventList[i].timeStamp;
                        break;
                }
        }

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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