



Telemotive Online Streaming Library 1.3.7

User's manual

Generated by Doxygen 1.8.0

Wed Dec 7 2016 13:08:39

# Inhaltsverzeichnis

<b>1</b>	<b>User's manual - Telemotive Online Streaming Library 1.3.7</b>	<b>1</b>
1.1	General	1
1.2	Compiler/Linker	1
1.3	Performance	1
1.4	Demo project	2
<b>2</b>	<b>Class Index</b>	<b>4</b>
2.1	Class List	4
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	IBPNGStreaming Struct Reference	5
3.2	IBPNGStreamingListener Struct Reference	10
	<b>Index</b>	<b>12</b>

## Kapitel 1

# User's manual - Telemotive Online Streaming Library 1.3.7

### 1.1 General

This is the documentation for the C++ Telemotive Online Streaming library which is compatible with all Microsoft compilers. The library's interface class [IBPNGStreaming](#) 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. *IChannel*, see *BPNGDefines.h*). The received trace data is forwarded to the application via listener callbacks (see [IBPNGStreamingListener](#)). Errors are processed by the functions' return values or via the appropriate listener callback (see section Error handling for more details).

#### 1.1.1 Error handling and listener mechanism

Errors are processed by the functions' return values or the listener callback function [IBPNGStreamingListener::onStreamingError\(\)](#). If the return value states an error a call to `getLastError()` provides details about the error occurred. Warnings are not intended to abort a process.

### 1.2 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.

### 1.3 Performance

There are currently no reliable measurments regarding the performance of the library.

## 1.4 Demo project

The "sample" directory contains a demo project for the Telemotive Online Streaming library.

Example:

```
/*
Sample project for BPNGOnlineStreamingLib (blue PiraT 2 Onlinestreaming Library)
*/
#include <iostream>
#include <conio.h>

#include "DataStreamingListener.h"
#include "IBPNGStreaming.h"
#include "BPNGDefines.h"

#include <time.h>

using namespace std;

int main()
{
    // 1. Set logger IP address
    const string ipAddress = "192.168.0.233";

    // 2. Create onlinestreaming interface
    int timeOutInMilliSec = 0;
    IBPNGStreaming* streaming = getBPNGStreamingInterface(timeOutInMilliSec, true);
    if (!streaming)
    {
        cout << "Failed to create pointer to OnlineStreamingInterface" << endl;
        return -1;
    }

    // 3. Create data streaming listener
    bool writeDataToFile = false;
    DataStreamingListener dataStreamListener(writeDataToFile);
    streaming->addStreamingListener((IBPNGStreamingListener*)&
        dataStreamListener);

    // 4. Disconnect interface from logger
    if (!streaming->connectLogger(ipAddress.c_str(), false))
    {
        releaseBPNGStreamingInterface(streaming);
        return -1;
    }

    // 5. Select channels you want to stream
    IChannelList* channelList = streaming->getChannelList();
    if(channelList->getSize() == 0)
    {
        cout << "Did not receive list of streaming channels from interface" << endl;
        releaseBPNGStreamingInterface(streaming);
        return -1;
    }

    for (unsigned int i = 0; i < channelList->getSize(); i++)
    {
        const IChannel* channel = channelList->getChannel(i);
        // e.g. only select channel CAN #1. Note that channel index always start at zero
        if (channel->getType() == CH_CAN && channel->getIndex() == 0)
        {
            streaming->setChannelFilter(channel->getType(), channel->getIndex(), true);
        }
    }

    cout << "Press any key to start streaming" << endl;
    _getch();
    time_t now = time(0);
    cout << "Started at " << asctime(gmtime(&now)) << endl;

    // 6. Start streaming data
    if(!streaming->startDataStreaming())
    {
        BPNGError err = streaming->getLastLastError();
        cout << err.msg << endl;
        releaseBPNGStreamingInterface(streaming);
        return -1;
    }
}
```

```
}

cout << "Press any key to stop streaming" << endl;
_getch();

// 7. Stop streaming data
if(!streaming->stopDataStreaming())
{
    BPNGError err = streaming->getLastError();
    cout << err.code << endl;
    cout << err.msg << endl;
    releaseBPNGLStreamingInterface(streaming);
    return -1;
}

// 9. Disconnect interface from logger
streaming->disconnectLogger();
releaseBPNGLStreamingInterface(streaming);
cout << "Streaming stopped, logger disconnected" << endl;
cout << "Good bye (press any key)" << endl;
_getch();
return 0;
}
```

## Kapitel 2

# Class Index

### 2.1 Class List

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

<a href="#">IBPNGStreaming</a>	Interface for the blue PiraT 2 streaming library . . . . .	5
<a href="#">IBPNGStreamingListener</a>	The BPNGStreamingListener struct Listener receives data from interface in const char* format Please implement a child class from this struct to handle incoming messages . . . . .	10

## Kapitel 3

# Class Documentation

### 3.1 IBPNGStreaming Struct Reference

Interface for the blue PiraT 2 streaming library.

```
#include <IBPNGStreaming.h>
```

#### Public Member Functions

- virtual BOOL WINAPI [connectLogger](#) (const char \*ipAddr, bool keepAlive=true)=0  
*Connect this interface to ipAddr. keepAlive set to true will prevent to logger from going down in case of no data traffic.*
- virtual void WINAPI [disconnectLogger](#) ()=0  
*Disconnect logger currently connected to this interface. The streaming process will be stopped.*
- virtual IChannelList \*WINAPI [getChannelList](#) ()=0  
*Returns a list of all available logger channels.*
- virtual ITesttoolsChannelList \*WINAPI [getTesttoolsChannelList](#) ()=0  
*Returns a list of all available logger testtools channels.*
- virtual BOOL WINAPI [isChannelStreamable](#) (ChannelType type, int channelIndex)=0  
*Check whether a channel is streamable.*
- virtual BOOL WINAPI [setChannelFilter](#) (ChannelType type, int channelIndex, BOOL activate)=0  
*Activate or deactivate a channel for streaming.*
- virtual void WINAPI [addCANIdFilter](#) (uint8\_t canChannelIndex, uint32\_t id)=0
- virtual void WINAPI [removeCANIdFilter](#) (uint8\_t canChannelIndex, uint32\_t id)=0  
*Remove a CAN-ID from the whitelist of a CAN channel.*
- virtual void WINAPI [clearCANIdFilters](#) (int8\_t canChannelIndex)=0  
*Clears the whitelist of a CAN channel. Passing -1 will clear all CAN channels.*
- virtual void WINAPI [addFlexRayFrameIdFilter](#) (uint8\_t flexrayChannelIndex, uint32\_t id, uint32\_t baseCyle, uint32\_t repet)=0
- virtual void WINAPI [removeFlexRayFrameIdFilter](#) (uint8\_t flexrayChannelIndex, uint32\_t id, uint32\_t baseCyle, uint32\_t repet)=0  
*Remove a frame ID (slot ID) from the whitelist of a FlexRay channel.*

- virtual void WINAPI [clearFlexRayFrameldFilters](#) (int8\_t flexrayChannelIndex)=0  
*Clears the whitelist of a FlexRay channel. Passing -1 will clear all FlexRay channels.*
- virtual void WINAPI [addLINIdFilter](#) (uint8\_t linChannelIndex, uint32\_t id)=0
- virtual void WINAPI [removeLINIdFilter](#) (uint8\_t linChannelIndex, uint32\_t id)=0  
*Remove a LIN-ID from the whitelist of a LIN channel.*
- virtual void WINAPI [clearLINIdFilters](#) (int8\_t linChannelIndex)=0  
*Clears the whitelist of a LIN channel. Passing -1 will clear all LIN channels.*
- virtual void WINAPI [addMOSTCtrlFilter](#) (int16\_t sourceAddr, int16\_t targetAddr)=0  
*Add a MOST filter. Pass -1 for arbitrary addresses. For an empty whitelist all messages will be passed through.*
- virtual void WINAPI [removeMOSTCtrlFilter](#) (int16\_t sourceAddr, int16\_t targetAddr)=0  
*Remove a MOST filter from CTRL .*
- virtual void WINAPI [clearMOSTCtrlFilters](#) ()=0  
*Remove all MOST filters.*
- virtual void WINAPI [addMOSTDataFilter](#) (int16\_t sourceAddr, int16\_t targetAddr)=0  
*Add a MOST filter. Pass -1 for arbitrary addresses. For an empty whitelist all messages will be passed through.*
- virtual void WINAPI [removeMOSTDataFilter](#) (int16\_t sourceAddr, int16\_t targetAddr)=0  
*Remove a MOST filter from CTRL .*
- virtual void WINAPI [clearMOSTDataFilters](#) ()=0  
*Remove all MOST filters.*
- virtual void WINAPI [addEthernetTextFilter](#) (int ethernetChannelIndex, const char \*filterText)=0  
*Add a Ethernet filter. Ethernet messages including the passed filter text will be passed through.*
- virtual void WINAPI [removeEthernetTextFilter](#) (int ethernetChannelIndex, const char \*filterText)=0  
*Remove a ethernet text filter.*
- virtual void WINAPI [clearEthernetFilters](#) (int ethernetChannelIndex)=0  
*Remove all ethernet filters.*
- virtual void WINAPI [addSerialTextFilter](#) (int serialChannelIndex, const char \*filterText)=0  
*Add a serial filter. Serial messages including the passed filter text will be passed through.*
- virtual void WINAPI [removeSerialTextFilter](#) (int serialChannelIndex, const char \*filterText)=0  
*Remove a serial text filter.*
- virtual void WINAPI [clearSerialFilters](#) (int serialChannelIndex)=0  
*Remove all serial filters.*
- virtual BOOL WINAPI [startDataStreaming](#) ()=0  
*Start streaming on all activated channels.*
- virtual BOOL WINAPI [stopDataStreaming](#) ()=0  
*Stop streaming.*
- virtual void WINAPI [useLoggerTimeZone](#) (BOOL flag)=0
- virtual BOOL WINAPI [assignDBCFile](#) (uint8\_t canChannelIndex, const char \*dbcFilePathAndName)=0  
*Assign dbc file to a CAN channel.*
- virtual void WINAPI [resolveCANIDs](#) (BOOL flag)=0  
*Set whether to resolve CAN IDs to message names from DBC files.*
- virtual BPNGError [getLastError](#) ()=0  
*Returns the last error occurred while online streaming.*



- virtual const char \* [getLoggerIp](#) ()=0  
*Returns the IP address of the logger currently connected.*
- virtual const char \* [getLoggerName](#) ()=0  
*Returns the name of the logger currently connected.*
- virtual const char \*WINAPI [getLibVersion](#) ()=0
- virtual const char \*WINAPI [getTMAAsciiVersion](#) ()=0
- virtual void [setTimeOut](#) (unsigned int timeOutMilliSec)=0
- virtual void WINAPI [addStreamingListener](#) (IBPNGStreamingListener \*listener)=0  
*Add a listener to the interface.*
- virtual void WINAPI [removeStreamingListener](#) (IBPNGStreamingListener \*listener)=0  
*Remove a listener from the interface.*
- virtual uint64\_t WINAPI [getTimeOfChannel](#) (ChannelType type, uint8\_t index)=0  
*Return starttime of particular channel.*
- virtual bool WINAPI [isRunning](#) ()=0  
*Return current state of streaming.*
- virtual int WINAPI [getActiveChannelCount](#) ()=0  
*Return the number of channels that are streaming.*
- virtual BOOL WINAPI [setMOST150MessageBufferSize](#) (int size)=0  
*Set the buffer size for MOST data, max is 41943040 Byte (40MB)*
- virtual BOOL WINAPI [setEthernetMessageBufferSize](#) (int size)=0  
*Set the buffer size for Ethernet data, max is 41943040 Byte (40MB)*

### 3.1.1 Detailed Description

Interface for the blue PiraT 2 streaming library.

To get access to a blue PiraT 2 data logger you need a pointer to an implementing instance of [IBPNGStreaming](#). Use [getBPNGStreamingInterface\(\)](#) to get such a pointer. This will create an instance on the heap, which has to be deleted with [releaseBPNGStreamingInterface\(\)](#) when not needed any more. Don't call the delete operator directly on this pointer. This interface allows your application to stream data on several channels from the blue PiraT 2 logger in realtime.

```

/*
Sample project for BPNGOnlineStreamingLib (blue PiraT 2 Onlinestreaming Library)
*/
#include <iostream>
#include <conio.h>

#include "DataStreamingListener.h"
#include "IBPNGStreaming.h"
#include "BPNGDefines.h"

#include <time.h>

using namespace std;

int main()
{
    // 1. Set logger IP address
    const string ipAddress = "192.168.0.233";

    // 2. Create onlinestreaming interface
    int timeOutInMilliSec = 0;
    IBPNGStreaming* streaming = getBPNGStreamingInterface(timeOutInMilliSec, true);
    if (!streaming)
    {
        cout << "Failed to create pointer to OnlineStreamingInterface" << endl;
        return -1;
    }
}

```

```

// 3. Create data streaming listener
bool writeDataToFile = false;
DataStreamingListener dataStreamListener(writeDataToFile);
streaming->addStreamingListener((IBPNGStreamingListener*)&
    dataStreamListener);

// 4. Disconnect interface from logger
if (!streaming->connectLogger(ipAddress.c_str(), false))
{
    releaseBPNGStreamingInterface(streaming);
    return -1;
}

// 5. Select channels you want to stream
IChannelList* channelList = streaming->getChannelList();
if(channelList->getSize() == 0)
{
    cout << "Did not receive list of streaming channels from interface" << endl;
    releaseBPNGStreamingInterface(streaming);
    return -1;
}

for (unsigned int i = 0; i < channelList->getSize(); i++)
{
    const IChannel* channel = channelList->getChannel(i);
    // e.g. only select channel #1. Note that channel index always start at zero
    if (channel->getType() == CH_CAN && channel->getIndex() == 0)
    {
        streaming->setChannelFilter(channel->getType(), channel->getIndex(), true);
    }
}

cout << "Press any key to start streaming" << endl;
_getch();
time_t now = time(0);
cout << "Started at " << asctime(gmtime(&now)) << endl;

// 6. Start streaming data
if(!streaming->startDataStreaming())
{
    BPNGError err = streaming->getLastError();
    cout << err.msg << endl;
    releaseBPNGStreamingInterface(streaming);
    return -1;
}

cout << "Press any key to stop streaming" << endl;
_getch();

// 7. Stop streaming data
if(!streaming->stopDataStreaming())
{
    BPNGError err = streaming->getLastError();
    cout << err.code << endl;
    cout << err.msg << endl;
    releaseBPNGStreamingInterface(streaming);
    return -1;
}

// 9. Disconnect interface from logger
streaming->disconnectLogger();
releaseBPNGStreamingInterface(streaming);
cout << "Streaming stopped, logger disconnected" << endl;
cout << "Good bye (press any key)" << endl;
_getch();
return 0;
}

```

### 3.1.2 Member Function Documentation

**3.1.2.1** `virtual void WINAPI IBPNGStreaming::addCANIdFilter ( uint8_t canChannelIndex, uint32_t id )`  
`[pure virtual]`

Add a CAN-ID to the whitelist of a CAN channel. Only IDs on the whitelist will be passed through. For an empty whitelist all messages will be passed through.

**3.1.2.2** `virtual void WINAPI IBPNGStreaming::addFlexRayFrameIdFilter ( uint8_t flexrayChannelIndex, uint32_t id, uint32_t baseCyle, uint32_t repet )`  
`[pure virtual]`

Add a frame ID (is a combination of slot ID, baseCycle and repetition) to the whitelist of a Flex-Ray channel. Only frame IDs on the whitelist will be passed through. For an empty whitelist all messages will be passed through.

**3.1.2.3** `virtual void WINAPI IBPNGStreaming::addLINIdFilter ( uint8_t linChannelIndex, uint32_t id )`  
`[pure virtual]`

Add a LIN ID to the whitelist of a LIN channel. Only IDs on the whitelist will be passed through. For an empty whitelist all messages will be passed through.

**3.1.2.4** `virtual BOOL WINAPI IBPNGStreaming::isChannelStreamable ( ChannelType type, int channelIndex )`  
`[pure virtual]`

Check whether a channel is streamable.

Channels that are deactivated in the logger's configuration can not be streamed. Ethernet channels with configured spy mode are also not streamable.

#### Parameters

<i>type</i>	The channel type of the channel to be checked for streaming support
<i>channelIndex</i>	The channel index of the channel to be checked for streaming support

#### Returns

1 = channel is streamable, 0 = channel is not streamable

**3.1.2.5** `virtual BOOL WINAPI IBPNGStreaming::setChannelFilter ( ChannelType type, int channelIndex, BOOL activate )`  
`[pure virtual]`

Activate or deactivate a channel for streaming.

Call this function for each channel you want to stream. A list of available channels can be retrieved by [getChannelList\(\)](#). Note that only those channels can be streamed, that are activated in the configuration of the blue PiraT 2 data logger. You can check whether the channel can be streamed with the function [isChannelStreamable\(\)](#). Note that filters can only be changed while streaming is stopped.

#### Parameters

<i>type</i>	The channel type of the channel to be activated or deactivated
<i>channelIndex</i>	The channel index of the channel to be activated or deactivated
<i>activate,:</i>	1 = activate, 0 = deactivate

**Returns**

1 = streaming filter is successfully set, 0 = streaming filter could not be set

**See Also**

[getChannelList](#), [isChannelStreamable](#)

**3.1.2.6 virtual BOOL WINAPI IBPNGStreaming::startDataStreaming ( ) [pure virtual]**

Start streaming on all activated channels.

Streaming can only be started once. If streaming is already started, the function will return true.

**Returns**

0 on failure, 1 on success

**3.1.2.7 virtual void WINAPI IBPNGStreaming::useLoggerTimeZone ( BOOL flag ) [pure virtual]**

Specifies whether to use the logger time zone for time stamp conversion. Default is UTC. Takes effect with next call of [startDataStreaming\(\)](#).

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

- IBPNGStreaming.h

## 3.2 IBPNGStreamingListener Struct Reference

The BPNGStreamingListener struct Listener receives data from interface in const char\* format Please implement a child class from this struct to handle incoming messages.

```
#include <IBPNGStreaming.h>
```

**Public Member Functions**

- virtual void [onStreamingMessage](#) (const char \*msg, const unsigned int length, const char \*msgKey)=0  
*Callback to stream incoming messages.*
- virtual void [onStreamingError](#) (BPNGError err)=0  
*Callback to notify a streaming error, in case of error streaming is always stopped.*
- virtual void [onDataDiscarded](#) (int numBytes, IChannelList \*channels)=0  
*Callback to notify listener about discarded messages or data.*
- virtual BOOL [isASCIIRequested](#) ()=0  
*Defines how the listener wants to receive the streamed messages - as ASCII (return true) or as byte stream (return false)*

### 3.2.1 Detailed Description

The BPNGStreamingListener struct Listener receives data from interface in const char\* format. Please implement a child class from this struct to handle incoming messages.

### 3.2.2 Member Function Documentation

#### 3.2.2.1 virtual BOOL IBPNGStreamingListener::isASCIIRequested ( ) [pure virtual]

Defines how the listener wants to receive the streamed messages - as ASCII (return true) or as byte stream (return false)

The messages are forwarded to the [onStreamingMessage\(\)](#) listener function:

- ASCII: null terminated c-string in Telemotive ASCII Format.
- Byte array: as raw byte array in Telemotive Trace File Format (TMT)

Specifications of the formats are available at the Telemotive's support center.

#### 3.2.2.2 virtual void IBPNGStreamingListener::onDataDiscarded ( int numBytes, IChannelList \* channels ) [pure virtual]

Callback to notify listener about discarded messages or data.

On high data traffic the streaming library may not be able to process all incoming data. In that case the received raw data must be discarded. The *channels* parameter identifies the channels that are affected.

#### 3.2.2.3 virtual void IBPNGStreamingListener::onStreamingMessage ( const char \* msg, const unsigned int length, const char \* msgKey ) [pure virtual]

Callback to stream incoming messages.

If messages are streamed as byte array *msg* is a pointer to the messages's first byte and *length* is the byte array's length. Otherwise *msg* is a null terminated c string and you don't have to consider the length parameter. If you want to display messages in a fixed position mode, you can use the third parameter *msgKey* to identify the position (row) in your viewer that has to be updated.

#### Parameters

<i>msg</i>	Pointer to the message either as byte array or as c-string
<i>length</i>	the message's length (only relevant for byte arrays)
<i>updateKey</i>	key needed for fixed position display mode (not supported yet)
<i>address</i>	needed for separating different loggers in one network that may be configured equally!

#### See Also

[getBPNGStreamingInterface\(\)](#), [isASCIIRequested\(\)](#)

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

- IBPNGStreaming.h

# Index

- addCANIdFilter
  - IBPNGStreaming, [8](#)
- addFlexRayFrameIdFilter
  - IBPNGStreaming, [9](#)
- addLINIdFilter
  - IBPNGStreaming, [9](#)
- IBPNGStreaming, [5](#)
  - addCANIdFilter, [8](#)
  - addFlexRayFrameIdFilter, [9](#)
  - addLINIdFilter, [9](#)
  - isChannelStreamable, [9](#)
  - setChannelFilter, [9](#)
  - startDataStreaming, [10](#)
  - useLoggerTimeZone, [10](#)
- IBPNGStreamingListener, [10](#)
  - isASCIIRequested, [11](#)
  - onDataDiscarded, [11](#)
  - onStreamingMessage, [11](#)
- isASCIIRequested
  - IBPNGStreamingListener, [11](#)
- isChannelStreamable
  - IBPNGStreaming, [9](#)
- onDataDiscarded
  - IBPNGStreamingListener, [11](#)
- onStreamingMessage
  - IBPNGStreamingListener, [11](#)
- setChannelFilter
  - IBPNGStreaming, [9](#)
- startDataStreaming
  - IBPNGStreaming, [10](#)
- useLoggerTimeZone
  - IBPNGStreaming, [10](#)