



# API Design Document

---

*Real-Time NIRS Data Streaming SDK*

*NIRx Medical Technologies*

*Revision 2.0*

*8/15/2011*

Rev 2.0 (2011/08/15): Documentation of streaming data format corrected

## Contents

1.Introduction.....	3
2.System Overview.....	3
3.Prerequisites and Limitations.....	3
3.1System Requirements.....	3
3.2Limitations.....	4
4.Installation.....	4
5.Header Files.....	4
6.DLL Function Call Listing.....	4
1.initialize.....	5
2.close.....	5
3.connect.....	6
4.disconnect.....	7
5.getStatus.....	7
6.getChannels.....	8
7.getName.....	9
8.Start.....	10
9.Stop.....	12
10.getFramesAvail.....	12
11.getNFrames.....	13
12.util_getAPIVersion.....	16
13.util_getErrorMsg.....	16

## List of Figures

Figure 1: System Overview

## 1. Introduction

This document details the Application Programmer Interface (API) for remotely streaming data from the NIRx Tomography System over Ethernet to a client application using a 32-bit Windows C DLL.

## 2. System Overview

This remote data steaming system was designed in two parts, a network server incorporated into the existing Tomography System, and a C DLL for use with 3<sup>rd</sup> party, client applications. These two parts communicate using TCP over a standard Ethernet network.

Client applications can be developed in any programming language that supports the use of Win32 C DLLs (Dynamic Linked Libraries), including C, C++, MATLAB, LabVIEW, and many others. By incorporating TomographyAPI.dll, these client applications are given a simple interface to control data streaming without needing to understand the complexities of network communication or client-server protocols.

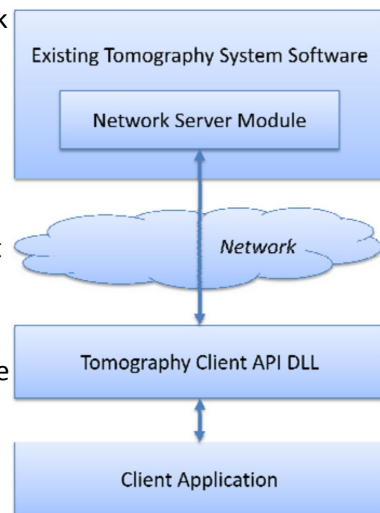


Figure 1: System Overview

## 3. Prerequisites and Limitations

### 3.1 System Requirements

In order to use this DLL within a third party application, the system must meet the following requirements:


- Standard desktop or laptop PC
- Using Microsoft Windows XP, Vista or 7 (only 32-bit editions of these operating systems are supported)
- Have the National Instruments LabVIEW 2009 Runtime Engine installed (note: the Tomography SDK Installer will automatically install the LabVIEW 2009 Runtime Engine as part of the installation process)
- Calling application (e.g. MATLAB or C compiler) must be a 32-bit process, or have the facility to call 32-bit DLLs.
- A operable TCP/IP connection between the client computer and tomography system must be in place.

## 3.2 Limitations

This software has been designed to give the user of the API a simple, robust interface. To support this goal the following limitations have been placed on the system:

- The network server will only support connections from a single client.
- A client can only connect to a single server.
- The network must have sufficient bandwidth to support the requested data transfer rates.
- There is no authentication for the Tomography system. This means that there is no software restrictions on the client computers that can connect to the system – any PC with the SDK installed and a valid network connection to the system will be able to stream data.

## 4. Installation

The Tomography System SDK can be installed using the supplied CD (or alternative media). Once the CD is inserted into the computer, click on “Start” and “Run...” on Windows XP or on Windows Vista and Windows,, hold down the Windows Key (“”) and press “R”. In the dialog that appears, type D:\Setup.exe (where D is the letter of your CD drive) and click on OK.

This setup program will install the Tomography System SDK to the desired location (normally C:\Program Files\NiRX Tomography SDK) and also install the required National Instruments LabVIEW 2009 Runtime Engine.

The DLL (TomographyAPI.dll) is then ready to use via the standard methods for the calling application.

## 5. Header Files

The included “C” Header file, TomographyAPI.h, details the function calls for the DLL and explains the different error codes used in the software development kit.

## 6. DLL Function Call Listing

Each function call exposed by TomographyAPI.dll is listed below, along with a brief description of the function and its inputs and outputs.

## 1. initialize

### 1.1. Summary

Description	Initializes the driver. Must be called before any other API functions (except <i>version</i> ).
Prototype	int initialize(void)

### 1.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

## 2. close

### 2.1. Summary

Description	Closes the driver, stops the network connection thread, and frees up any memory in use. This should be called when the API is no longer needed.
Prototype	int close(void)

### 2.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

### 3. connect

#### 3.1. Summary

Description	Connects to the server at a specified address and port.
Prototype	int connect(char address[], unsigned short tcpPort, int timeout)

#### 3.2. Parameters

##### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

##### *address*

Direction	Input
Data Type	C-string
Description	IP Address or hostname of target Tomography System. Memory for this C-String should be allocated prior to calling this function.

##### *tcpPort*

Direction	Input
Data Type	unsigned short
Description	TCP port for sending command messages and receiving streamed data. This should match the configuration of the Tomography System's network server.

##### *timeout*

Direction	Input
Data Type	int
Description	Network timeout in milliseconds (-1 for unlimited.) The system will wait for a response for this specified time before generating a timeout error.

## 4. disconnect

### 4.1. Summary

Description	Closes the active connection with the server.
Prototype	int disconnect(void)

### 4.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

## 5. getStatus

### 5.1. Summary

Description	Gets the status of the API
Prototype	int getStatus(unsigned int *statusFlags, double *sampleRate)

### 5.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

#### *statusFlags*

Direction	Output
Data Type	unsigned int*
Description	A bit-field value describing the status of the system. See TomographyAPI.h for individual bit assignments.

### *sampleRate*

Direction	Output
Data Type	double*
Description	The published sample rate of the system.

## 6. getChannels

### 6.1. Summary

Description	Gets the number of channels available on the system.
Prototype	int getChannels(int *sources, int *detectors, int *wavelengths)

### 6.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

#### *sources*

Direction	Output
Data Type	int*
Description	The number of sources available on the system

#### *detectors*

Direction	Output
Data Type	int*
Description	The number of detectors available on the system



### *wavelengths*

Direction	Output
Data Type	int*
Description	The number of wavelengths available on the system

## 7. getName

### 7.1. Summary

Description	Gets the name of the associated Source / Detector / Wavelength.
Prototype	int getName(unsigned short nameType, int index, char pName[], int nameLength);

### 7.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

### *nameType*

Direction	Input
Data Type	unsigned short
Description	An enumerated value for the type of name to be received (0=Source, 1=Detector, 2=Wavelength)

### *index*

Direction	Input
Data Type	int
Description	The source/detector/wavelength index whose name should be returned (zero-based, so 0=1 <sup>st</sup> S/D/W, 1=2 <sup>nd</sup> , etc)

### *pName*

Direction	Output
Data Type	C-String
Description	The requested channel's name. Memory for this C-String should be allocated prior to calling this function.

### *nameLength*

Direction	Input
Data Type	int
Description	The size of the buffer previously allocated for the pName output. The buffer should be large enough to contain the requested channel name, as set in the Tomography System Network Server. If the name is longer than the buffer (or this value), the returned pName will be truncated.

## 8. Start

### 8.1. Summary

Description	Asks the server to start sending data frames for a given set of channels. Providing NULL pointers and associated sizes of 0 will signify that all Sources / Detectors / Wavelengths will be transmitted.
Prototype	int start(int sources[], int detectors[], int wavelenths[], int numSources, int numDetectors, int numWavelengths, int *elementsPerFrame)

### 8.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

### *sources*

Direction	Input
Data Type	int[]
Description	An array of the source indexes to be streamed, a zero-length array signifies “all”

### *detectors*

Direction	Input
Data Type	int[]
Description	An array of the detector indexes to be streamed, a zero-length array signifies “all”

### *wavelengths*

Direction	Output
Data Type	int[]
Description	An array of the wavelength indexes to be streamed, a zero-length array signifies “all”

### *numSources*

Direction	Input
Data Type	int
Description	The size of the supplied sources array.

### *numDetectors*

Direction	Input
Data Type	int
Description	The size of the supplied sources array.

### *numWavelengths*

Direction	Input
Data Type	int
Description	The size of the supplied sources array.

### *elementsPerFrame*

Direction	Output
Data Type	int*
Description	The number of elements in the array returned by <i>getNFrames</i> that correspond to a single frame. See <i>getNFrames</i> for more information.

## 9. Stop

### 9.1. Summary

Description	Asks the server to stop sending data
Prototype	int stop(void)

### 9.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

## 10. getFramesAvail

### 10.1. Summary

Description	Get the number of frames available in the local machine's buffer.
Prototype	getFramesAvail(int *frameCount)

## 10.2. Parameters

### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

### *frameCount*

Direction	Output
Data Type	int*
Description	The number of data frames in the local buffer.

## 11. getNFrames

### 11.1. Summary

Description	Get the next N available frames (or timeout). If the number of frames requested is not yet available, this function will wait until either the required number of frames becomes available or the timeout time is reached.
Prototype	getNFrames(int reqFrames, int timeout, int *frameCount, double timestamps[], char timingBytes[], float data[], int *dataBufferSize)

### 11.2. Parameters

#### *Return Value*

Direction	Output
Data Type	int
Description	Corresponds to the error code of the system – see TomographyAPI.h for more information.

### *reqFrames*

Direction	Input
Data Type	int
Description	The number of data frames requested

### *timeout*

Direction	Input
Data Type	int
Description	The maximum time in milliseconds that the function call should wait for if the requested number frames are not yet available. If this value is -1, this function will wait forever until all of the requested frames are available. If it is 0, this function will return the number of frames requested or the number currently in the local buffer, whichever is smaller.

### *frameCount*

Direction	Output
Data Type	Int*
Description	The actual number of frames returned

### *timestamps*

Direction	Output
Data Type	double[]
Description	The number of seconds since 1/1/1904 00:00:00 UTC (Gregorian calendar, ignoring leap seconds) associated with each frame. This array will be of length <i>frameCount</i> .

### *timingBytes*

Direction	Output
Data Type	char[]
Description	The timing byte value associated with each frame. This array will be of length <i>frameCount</i> .

### *data*

Direction	Output
Data Type	float[]
Description	A 1D array containing the data for the returned frames. This data can be parsed using the format displayed in <i>Frame Data Structure</i> , below. Memory should be allocated for this array prior to calling this function. The maximum number of elements this array could contain will be the number of requested frames multiplied by the number of array elements per frame (see <i>start</i> for <i>elementsPerFrame</i> )

### *dataBufferSize*

Direction	Input / Output
Data Type	Int*
Description	The number of elements in the pre-allocated data array. Following completion of this function call, this value will be replaced with the actual number of elements returned.

## 11.3. Frame Data Structure

The number of elements in the data 1D array that correspond to a single frame are:

$$\text{sources} * \text{detectors} * \text{wavelengths},$$

and is also returned in the *start* function as the *elementsPerFrame* parameter. As such, the data array will be

$$N * \text{elementsPerFrame}$$

where N is the number of frames read.

Within each *elementsPerFrame* elements, the data is structured along the following lines:

<S1,D1,W1>,<S1,D2,W1>...<S1,Dmax,W1>,<S2,D1,W1>,<S2,D2,W1>...<S2,Dmax,W1>...<Smax,Dmax,W1>,  
<S1,D1,W2>,<S1,D2,W2>...<S1,Dmax,W2>,<S2,D1,W2> ...<Smax,Dmax,Wmax>

Where S1, S2, etc are the Sources, D1, D2, etc are the Detectors, and W1, W2, etc are the wavelengths.

## 12. util\_getAPIVersion

### 12.1. Summary

Description	This utility function returns the version number of the API
Prototype	void util_getAPIVersion(versionStruct *APIVersion)
Return Value	None

### 12.2. Parameters

#### *APIVersion*

Direction	Output
Data Type	versionStruct* (see TomographyAPI.h for more information)
Description	The version number of the API in <Major>.<Minor>.<Fix>.<Build> notation

## 13. util\_getErrorMsg

### 13.1. Summary

Description	This utility function can be used to get a description for the error codes returned by the other functions.
Prototype	void util_getErrorMsg(int errorCode, char errorMessage[], int *messageLength)

### 13.2. Parameters

#### *errorCode*

Direction	Input
Data Type	int
Description	The error code whose message is being queried

#### *errorMessage*

Direction	Output
Data Type	C-String
Description	The message describing the supplied error code. The memory for this C-String must be pre-allocated before this function is called.



### *messageLength*

Direction	Input
Data Type	int
Description	The size of the C-String pre-allocated for errorMessage. If the error message is longer than this value, it will be truncated.