Delsys Application Program Interface

User's Guide

MAN-033-1-2

TABLE OF CONTENTS

1 Important Information

1.1. Intended Use

The Delsys API is a software development tool to be used in conjunction with the Trigno Wireless Biofeedback System. The API is not intended to perform medical assessments or diagnostic procedures. It is intended to be used as a software component of a third-party software application. The function of the API is to manage the transfer of data from the Trigno System to third-party software applications, and is designed to work exclusively with the Trigno System. It is designed to facilitate communication with the Trigno System from a third-party software application. Any claims regarding the intended use of the 3rd party software are the sole responsibility of the 3rd party software manufacturer.

1.2. Technical Service and Support

For information and assistance, please visit:
www.delsys.com

Contact us:
E-mail: support@delsys.com
Telephone: (508) 545 8200

1.3. Device Information

Please see the Trigno Wireless Biofeedback System User Guide for information on the Trigno device.

1.4. System Requirements

- Windows 10 and above
- Microsoft Visual Studio 2017 or later

2 Delsys API Overview

The Delsys API is a .NET cross platform library created by Delsys with data collection in mind. The API is a tool allowing developers to create applications built on top of Delsys Hardware Technologies. More specifically, the Delsys API allows users to connect, configure, and collect data from Delsys Trigno Sensors.

This guide will give a technical overview of the modules making up the API, and include information on how to properly utilize the functionality of the API for an RF (base station) setup.

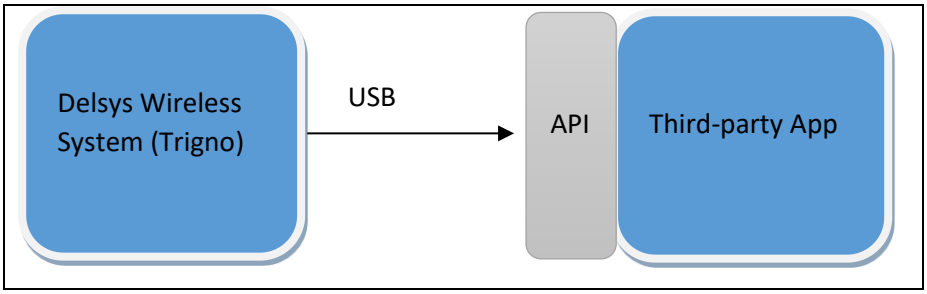

Figure 1: Data flow and API, Trigno System, and third-party app.

3 Using the API

Prior to beginning development with the Delsys API, please read the Trigno Wireless Biofeedback System User Guide (MAN-012) for information about using the Trigno hardware and its intended uses. It is also recommended that developers download and install the Trigno Discover Software Package and familiarize themselves with it to learn how the system is meant to be interacted with.

To use the Trigno System via the API, the third-party software application must perform these basic steps:

- Connect to the Trigno System.
- Configure the hardware (pair sensors, scan for sensors, etc.)
- Start/Stop the Trigno System.
- Trigger the system (optional). Send a start trigger to the Trigno Base Station. For more information on this please see the Trigno Wireless Biofeedback System User Guide (MAN-012).
- Receive Data

For a short, practical tutorial on how to perform all of those steps, please follow the API Quick Start Guide (MAN-032).

4 Definitions

4.1 Pipeline

Defines the flow of data from a single Data Source, ending with data being presented to the user. For example, in RF mode a pipeline would collect data from Trigno Hardware, process them, and make them available to the user. The pipeline contains an internal state machine which controls the flow of data.

The pipeline contains various useful events which can be used to collect data, and receive signals on system attributes.

Each pipeline exposes its own Component Manager and Transform manager (see details below).

4.2 Pipeline Controller

Responsible for coordinating and monitoring all available pipelines. **Note:** Delsys API currently supports only one pipeline inside the pipeline controller.

4.3 Pipeline States

Pipeline states will be available for viewing, and give developers a chance to see at what point in the data collection stage they are. The most important states are as follow

State	**Description**
Off	No sensors have been detected and subsystem is un-configured
Connected	Sensors are detected, subsystem configured. Sensors and transforms un-configured
InputsConfigured	Data source configuration has been finalized, as has the configuration for each component
OutputsConfigured	Data output configuration set. Transform channels set
Armed	All configurations have been finalized. Ready for data streaming
Running	Data collection is underway
Finished	Data collection is complete

4.4 Component

Defines a hardware device from which data is collected. A component will contain a variety of attributes, as well as specific data channels.

An example of a component would be a Trigno Sensor, which has a set of configuration options, as well as a defined number of channels (EMG, ACC, etc.) depending on the mode the sensor is operating in.

4.5 Channel

Defines a singular data stream, which is sent at regular intervals from a component. Channels have specific units (mV,g,etc.).

4.6 Data Source

Defines the hardware platform by which component data is aggregated. Example of this would be a Trigno Base Station. In order for the Delsys API to function, it must detect and be connected to a supported Data Source.

4.7 Component Manager

The Component Manager manages interaction with, and configuration of components. The Component Manager comes in two types, RF Manager and BT Manager. The component manager provides useful tools for interacting with components:

- Viewing and setting component configurations
- Viewing component attributes (available sample modes, current sample modes etc.).
- Viewing component channels.
- Viewing component properties (battery life, signal strength etc.).

4.8 Configurations

This section can be split up into 3 main configuration types. Each of these types defines how data will pass from the Data Source to the user. **Note:** configurations must always be applied before data can be collected.

4.8.1 *Data Source Configuration*

Defines attributes related to the Datasource. In RF mode, the configuration allows the Trigno Base to be set to specific configurations, as well as the setting and configuring of the Delsys Trigger Module.

4.8.2 *Component Configuration*

Defines how a component will behave during data collection. Components will be different depending on mode.

- Allow configuration of DIO pins.
- Allows user to select from a number of sampling/channel configurations and layouts. Sample modes may vary depending on sensor type.

4.8.3 *Output Configuration*

This configuration defines how data will appear to the user. The user will be able to select which Transform output channels they desire to view, and in what order. This allows for extreme flexibility in configuring the manner in which data is collected.

5 Technical Overview

The Delsys API uses these Objects to abstract away the hardware layer from Subscribers, in order to provide a robust but easy-to-use interface.

5.1 Components

At the highest level, components can be thought of as virtual collections of channels with specific properties and subroutines to modify data within those channels. A component can be formed in a variety of ways.

The Component list will represent a dynamic collection of components, providing the user with access to the following.

- Type Information: Attributes relative to the sensor types.
- Component Data: Synchronized chunks of acquired data passed through a series of data transformation steps.
- Channel Information: Channel structure, setup, attributes.

5.1.1 *Component Types*

Component Types are groupings of “bound” channels, and as such are able to be created, and extended on the fly. These channels may or may not belong to discrete devices, such as Trigno Sensors, or they could be a combination of discrete devices.

There are specific preset types of components created for the first production version of the API.

5.1.1.1 SensorTrignoRf

This class is derived from the Component base class, and serves as a base class for all Trigno Sensor types.

5.1.1.2 Trigno Sensor Types

Each Trigno Sensor type is a class derived from the SensorTrignoRf class. They are as follows:

Sensor Type	**# Type**
Avanti Sensor	14

Quattro	16
Galileo	17
Snap Lead Sensor	18
Spring Contact	19
FSR Adapter	20
EKG	21
Load Cell Adapter	22
Goniometer Adapter	23
Mini	24
Analog Input Adapter	25
Duo	27
Maize	29

5.1.2 *Trigno Sensor Properties and Methods*

Each instantiation of a Trigno Sensor as a component contains properties and methods which are used to configure the sensor and to inform the consumer of certain sensor properties.

5.1.3 *Component Channels*

Channels form the building blocks out of which Components are defined, with each channel defining the properties of a specific stream of data, along with the parameters that govern said stream. Channels are divided into the following hierarchy, which covers a wide variety of current types, while also being flexible and allowing for extensions and new channel types to be created and added.

5.1.3.1 Digital Channels

Will represent the channels of a digital device. These channels will not require operations such as voltage scaling and offset calculations.

5.1.3.2 Analog Channels

These channels are voltage based and may need to have scaling and offsets applied. Currently most smart sensor channels, especially EMG, fall under this category.

5.2 Pipeline Controller

Instance of the Pipeline Control module. Consuming applications are able to configure Data Sources, and control the Dataflow Pipeline from this instance.

5.2.1 *Collection Data Ready Event*

The data ready event is the main source by which a subscriber can get data after initiating a data collection. This event represents the output of the live data portion of the API pipeline. See section 6.1.1 & 6.1.13 for technical details.

5.3 Pipeline Controller Module (PCM)

The pipeline controller represents the control center for the API, issuing commands and interfacing with hardware devices.

Most importantly, the PCM is responsible for configuring Data Sources, which define the precise interaction between software buffers and data being received from hardware. The PCM will allow the API to access the hardware that will communicate with the API and supporting application. This process are abstracted to the user, with the exception of a few commands.

5.3.1 *Pipeline State Machine*

To collect data, the PCM is responsible for managing the data collection pipeline, with data undergoing various transformations. A state machine allows the PCM to transition the pipeline to various states of activity. In this way the PCM is able to go from initialization, transitioning until data collection is achieved.

5.3.2 *Input Configurations*

Subscribers wishing to utilize the API pipeline must specify an input configuration. A configuration consists of collections of properties that define all stages of DataSource functionality. Without specifying a configuration, a subscriber will not be able to enter data collection mode.

5.3.3 *Output Configurations*

Subscribers wishing to utilize the API pipeline must specify an output configuration. The internal pipeline transforms input data into various forms, and the number of output streams may no longer correspond to the number of input channels. An output configuration is a mapping that allows the system to successfully package data for the user in an intelligible fashion.

5.3.4 *External Commands*

The PCM will receive external API commands issued by the consumer. These commands will configure, activate, deactivate, and start/stop data acquisition. It will set up any basic structures needed by the API such as the Component Manager, and the Transform Manager. These commands can include:

5.3.4.1 *Command List*

1. Get Datasource
2. Configure Datasource
3. Get Components
4. Transition Pipeline
5. Subscribe to DataReady event.
6. Set or unset triggers.

6 Programming the API

Essential methods and properties of the API are described below. The list is not exhaustive but contains information necessary for proper understanding of the fundamental flow and operation of the API.

6.1 Pipelines

The pipeline is the most fundamental object of the API. To simplify the calls described below, the following reference is made.

6.1.1 *Initialize Data Source and Pipeline*

The Datasource will have to be called using the DeviceSourcePortable Class. In order to access the device, you will need to license your API product. NOTE: Must have an active key/license provided to you by Delsys Inc.

C#

```
var deviceSourceCreator = new DeviceSourcePortable(key, license);

_deviceSource = deviceSourceCreator.GetDataSource(SourceType.TRIGNO_RF);

_deviceSource.Key = key;
_deviceSource.License = license;
```

After receiving the Datasource, the following commands allow us to set up a pipeline of RF data and access the Component and Transform Managers. Finally subscribe your method to the API event callbacks:

C#

```
try
{
   PipelineController.Instance.AddPipeline(_deviceSource);
}
catch (BaseDetectionFailedException e)
{
   return;
}

_pipeline = PipelineController.Instance.PipelineIds[0];

_pipeline.TrignoRfManager.ComponentAdded += ComponentAdded;
_pipeline.TrignoRfManager.ComponentLost += ComponentLost;
_pipeline.TrignoRfManager.ComponentRemoved += ComponentRemoved;
_pipeline.TrignoRfManager.ComponentScanComplete += ComponentScanComplete;

_pipeline.CollectionStarted += CollectionStarted;
_pipeline.CollectionDataReady += CollectionDataReady;
_pipeline.CollectionComplete += CollectionComplete;
```

API Event	Description
ComponentAdded	Raised when a sensor is paired.
ComponentLost	Raised when a sensor as lost connection to base station or tirgno lite.

ComponentRemoved	Raised when a component is removed in software.
ComponentScanComplete	Raised when Scan() operation is complete.
CollectionStarted	Raised when the collection has started.
CollectionDataReady	Raised every frame interval during a data collection and contains all of the sensor/channel data.
CollectionComplete	Raised when the collection has terminated.

6.1.2 *Pairing*

Sensors must be paired to the base station or Trigno Lite before they are visible via the scan.

C#

```
await _pipeline.TrignoRfManager.AddTrignoComponent(new CancellationToken(), sensorNumber);
```

6.1.3 *Scanning*

Scanning will find sensors and make them available for allocation and configuration within the API. A scan must occur before you can steam your sensors.

C#

```
await _pipeline.Scan();
```

6.1.4 *Configure Pipeline*

The following code demonstrates how to configure the pipeline for raw data output from all (scanned) sensors.

C#

```
DataLine dataLine = new DataLine(_pipeline);
dataLine.ConfigurePipeline();

_frameThroughput = PipelineController.Instance.GetFrameThroughput();

int totalChannels = 0;
foreach (var comp in _pipeline.TrignoRfManager.Components)
{
   totalChannels += comp.TrignoChannels.Count();
}

PipelineStatus = _pipeline.CurrentState.ToString();
SensorsConnected = _pipeline.TrignoRfManager.Components.Count();
```

6.1.5 *Data Source Information Dictionary*

There are several properties that can be extracted from the Pipeline. These properties are accessible via a string-to-string dictionary in the Pipeline. Each dictionary key and a description of its associated value

are listed in the table below. Accessing this information can be done by the call below (after a Pipeline has been set up), replacing **Key** with whatever key string you wish to query the value of.

C#

```
_pipeline.DataSourceInfo[Key];
```

Key	Value Description
"Base ID"	The base identification number.
"Hardware Name"	The name of the base as identified by Windows.
"Hardware Address"	The USB port address of the base.
"Firmware Version"	The firmware version of the base.

6.1.6 *Sensor Component Objects*

Sensors have a robust selection of settable properties and attributes, accessible by using a reference to a sensor such as the one below (which assumes at least one component has been allocated.)

C#

```
PipelineController.Instance.PipelineIds[0].TrignoRfManager.Components[0];
```

6.1.7 *Sensor Type*

The sensor's type (e.g 14 is an Avanti sensor.)

6.1.8 *Sensor Serial*

The unique Serial ID of the sensor.

6.1.9 *Sensor Firmware*

The firmware version of the sensor.

6.1.10 *Sensor Mode*

The configured mode of the sensor.

6.1.11 *Sensor Channels*

The sensor's channels each have properties that may be queried.

C#

```
int sensorType = sensor.Properties.Type;
int sensorSID = sensor.Properties.Sid;
string sensorFW = sensor.Properties.Fw;
```

```
int sensorMode = sensor.Configuration.SampleMode;
var myChan = sensor.TrignoChannels[0];
```

6.1.11.1 *Sampling Rate*

The sample rate is the quotient of the samples per frame and frame interval. All three of these properties can be retrieved.

6.1.11.2 *Units*

The units of the values being output by the channel

C#

```
int sampleSize = myChan.SamplesPerFrame;
float frameInterval = myChan.FrameInterval;
float sampleRate = myChan.SampleRate;
var units = myChan.Units;
```

6.1.12 *Allocation/Deallocation*

Allocating a sensor and deallocating a sensor is done via the Pipeline's TrignoRfManager.

C#

```
var componentManager = _pipeline.TrignoRfManager;
componentManager.SelectComponentAsync(mySensor);
componentManager.DeselectComponentAsync(mySensor);
```

6.1.13 *Data Collection*

Call the .Start() method on the RF pipeline to begin the data stream. Data will come in real-time via the CollectionDataReady API event.

C#

```
await _pipeline.Start();
```

Access collected data:

See section 6.1.1 above for details about subscribing to the API's CollectionDataReady event. This code snippet demonstrates the callback event from the application side, and how to parse and store the collection data.

C#

```
List<List<List<double>>> AllCollectionData = new List<List<List<double>>>();
```

```
public void CollectionDataReady(object sender, ComponentDataReadyEventArgs e)
{
   List<List<double>> data = new List<List<double>>();

   for (int k = 0; k < e.Data.Count(); k++)
   {
      for (int i = 0; i < e.Data[k].SensorData.Count(); i++)
      {
         for (int j = 0; j < e.Data[k].SensorData[i].ChannelData.Count(); j++)
         {
            data.Add(e.Data[k].SensorData[i].ChannelData[j].Data);
         }
      }

   }

   AllCollectionData.Add(data);
}
```

*Note: At the end of the collection the AllCollectionData class variable will contain all of the channel data from the data collection. Each item in the outermost List is each CollectionDataReady packet (List<List<double>> data). The middle List item is each channel, and the last List item (List<double>) is the data for that channel. Reset AllCollectionData in between data collections.

Stop Data Collection:

```
await _pipeline.Stop();
```

//optional: DisarmPipeline command. Note: To begin another data stream via the Start() command, the pipeline does not need to be Disarmed if you are intending to use the same sensor configuration for the next trial. However, the pipeline must be disarmed if you want to scan, pair, or change sensor modes.

//Important! – If you want to Disarm the pipeline after each data stream, we strongly suggest moving this command to the CollectionComplete callback event to avoid internal state errors.

```
await _pipeline.DisarmPipeline();
```

6.2 Transforms

The Delsys API offers a variety of transforms that can operate on the sensor data in real time. Transforms operate on individual channels or a collection of channels. These channels do not need to have the same type or come from the same sensor.

Ultimately the transform configuration will always produce an output configuration stored in the `OutputConfig`. This serves as a map for the Delsys API and will determine what data will be sent during CollectioNDataReady events. The `OutputConfig` is used with `ApplyOutputConfigurations` move the pipeline into a state that can run data collection.

6.2.1 ***Raw Data***

In the simplest case only the channels native to a sensor, known as the raw data channels, will be present in the data stream. The `TransformConnector` is used to access the `SetupTransforms()` method and returns the outputConfig necessary to complete the Arm Output step.

C#

```
TransformConnector transformConnector = new TransformConnector(_pipeline);
OutputConfig outputConfig = transformConnector.SetupTransforms();
```

6.3 References

6.3.1 *API Quick Start Guide*

A short introduction and example of how the API can be implemented. This is an RF example, and requires a base station to run. It will walk you through connecting and disconnecting the pipeline in addition to pairing, configuring, and getting data from sensors. It is recommended you begin by following this guide, and referencing the other documentation (this User Guide included) to gain a fuller understanding of the API.

6.3.2 *Examples*

The API comes with a number of samples demonstrating how to create transforms and create a data collection. These examples can be useful to get started.