Delsys Application Program Interface

User's Guide

MAN-033-1-1

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 EMG biofeedback system. The API is not intended to perform assessment 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.

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 EMG System User Guide for information on the EMG Device.

1.4. System Requirements

1.4.1. Windows RF Developer System Requirements:

- Windows 7 and above
- Microsoft Visual Studio 2015 or later

1.4.2. Windows BT Developer System Requirements

- Windows 10 Build 16299 and above
- For UWP: Microsoft Visual Studio 2017 or later

1.4.3. Android Developer System Requirements:

- Windows 7 and above (64 bit)
- Microsoft Visual Studio 2015 or later
- Xamarin Framework (included in Visual Studio)
- Android 6.0 and above.

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. The following modes are currently supported:

- RF Mode
  - Allows streaming of data from Trigno Sensors to a PC via Delsys' proprietary RF protocol, and the Trigno Base Station.
- Bluetooth Mode
  - Allows streaming of data from Trigno Sensors directly to a Bluetooth-enabled device:
    - Android Tablets and Phones
    - Microsoft Surfaces, Windows Phones, Laptops and more

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.

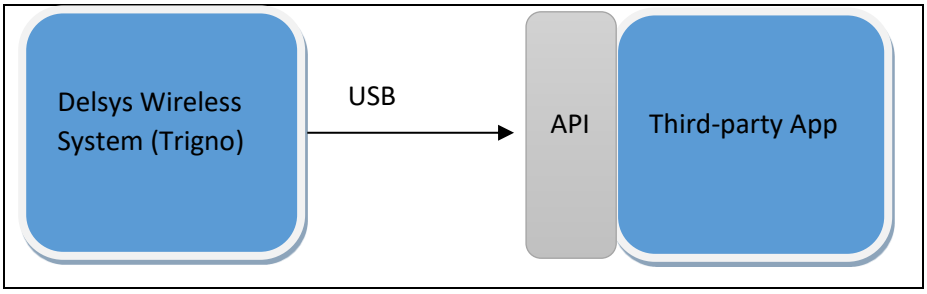

Figure 1: Data flow and API, Trigno System, and third-party app.

3 Using the API

Prior to beginning development with the Delsys API, please read the Trigno Wireless EMG 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 EMGworks 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, etc.). Please see the next section "Configuring the Trigno Hardware" for information on this.
- Start 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 EMG 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) included in the docs folder.

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 follows:

1. Off

   No sensors have been detected and subsystem is un-configured

2. Connected

   Sensors are detected, subsystem configured. Sensors and transforms un-configured.

3. Armed

   All configurations have been finalized.

4. Running

   Data collection is underway.

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.).

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, or an Android Tablet's Bluetooth subsystem. In order for the Delsys API to function, it must detect, and be connected to a supported Data Source.

4.7 Transform

A transform can be thought of as a "data filter". Data comes in one side via a defined number of input channels, is transformed in some manner, and goes out the other side via a defined number of output channels. What shape the transformation takes is defined by the transform applied.

Users will be able to apply a pre-existing transform, or define their own.

Note: the Delsys API currently supports only the default transform type (Raw Data).

4.8 Component Manager

The Component Manager manages interaction with, and configuration of components. The Component Manager comes in two flavors:

RF Manager:

- The Component Manager in RF mode.

BT Manager:

- The Component Manager in Bluetooth mode.

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.).

4.9 Transform Manager

Manages all Transforms and Transform Configurations. Allows users to view currently added transforms, add new transforms, and remove transforms.

4.10 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.

4.10.1 *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.

4.10.2 *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. These sampling modes may be different depending on whether operating in RF or Bluetooth mode. Sample modes may also vary depending on sensor type.

4.10.3 *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.

Components are presented as a generic collection, which are able to accept various component subsets. For example, if a Trigno Base has been connected, the list can be populated with connected Trigno Sensors. In the case of a generic DAQ device, the list may be populated with Components representing specific groupings of channels.

Many of the properties of the component list are open to view. Developers should be able to see many types of attributes and information about the component.

The Component List will also act as a configurable interface of the API. Users are able to set their own components and transforms that are bound to them.

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:

Class Name	Sensor Type
SensorStandardLegacy	0
SensorSpringContactLegacy	1
SensorSnapLeadLegacy	2
SensorStandardLegacy	3
SensorFSRLegacy	4
SensorEKGLegacy	5
SensorLoadCellLegacy	6
SensorGoniometerLegacy	7
SensorMiniLegacy	9
SensorAnalogInputLegacy	10
SensorImLegacy	11
SensorDRLegacy	12
SensorTriggerLegacy	13
SensorTrignoAvanti	14
SensorTrignoTemperature	15
SensorTrignoQuattro	16
SensorTrignoDEMG	17
SensorTrignoAvantiSnapLead	18

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 sink 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.

5.3 Slot Allocation

Slot allocation is the process by which we reorganize sensors in order to support multi-bandwidth modes without the need to repair sensors. Under the hood, we now have an additional step in configuration.

This step has three phases:

1. Create the optimal configuration of sensors based on which have been selected. For this we sort in descending order of bandwidth.
2. Send the base the configuration.
3. Enter the sensor assignment beacon state. For 10 seconds the base will negotiate with the sensors to ensure they are properly set into their slot(s). Sensors may blink teal.

This subprocess is rolled into the rest of our configuration, so no additional code must be written in order for this to occur.

5.3.1 ***Slot and Multi-Slot Sample Modes***

Slots are a division of the RF spectrum, optimized for our 16 sensor system, which both the base and the sensor use to stream data. As such, there is a maximum throughput for any one slot. Multi-slot functionality was developed to work around these bottlenecks, and push our sensors' capabilities even further. Using multiple slots we can multiplicatively increase the output of a sensor – making available higher sample rates, more channels, higher bit resolution, etc.

Due to RF interference and other considerations, the multi-slot allocations are not sequential or arbitrary. We currently are capable of supporting up to a 4-slot allocation for one sensor. Below is a table detailing the multi-slot groups (each group lists the slots the sensor occupies):

Dual Bandwidth (DBW)

1. 0, 4
2. 1, 5
3. 2, 6
4. 3, 7
5. 8, 12
6. 9, 13
7. 10, 14
8. 11, 15

Triple Bandwidth (TBW)

1. 0, 4, 8
2. 1, 5, 9
3. 2, 6, 10
4. 3, 7, 11

Quad Bandwidth (QBW)

1. 0, 4, 8, 12
2. 1, 5, 9, 13
3. 2, 6, 10, 14
4. 3, 7, 11, 15

Note: There can be at most four sensors in Triple Bandwidth mode.

5.3.2 ***Data Structures***

The API gives access to two data structures that represent the multi-slot configuration.

Flat Configuration Array

The flat sensor array is the 1-dimensional list of sensors and is what is eventually sent to the base for configuration. The matrix would look as such, where each number is the slot number associated with it. The value of an empty slot is -1.

Slot Associations in Array

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Example

Sensor in QBW mode with SID A is paired to slot 0, 4, 8, 12; Sensor in SBW mode with SID B is paired to slot 1; Sensor in SBW mode with SID C is paired to slot 6; Sensor in DBW mode with SID D is paired to slot 11, 15.

A	B	-1	-1
A	-1	C	-1
A	-1	-1	D
A	-1	-1	D

Logical Configuration Array

The logical configuration array is a 2-dimensional array used to optimize operations based on the idea of slot groups. The matrix would look as such, where each number is the slot number associated with it. The value of an empty slot is -1.

Slot Associations in Array

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Example

Sensor in QBW mode with SID A is paired to slot 0, 4, 8, 12; Sensor in SBW mode with SID B is paired to slot 1; Sensor in SBW mode with SID C is paired to slot 6; Sensor in DBW mode with SID D is paired to slot 11, 15.

A	A	A	A
B	-1	-1	-1
-1	C	-1	-1
-1	-1	D	D

5.4 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 are 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.

5.4.1 *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.

For more information about the Pipeline State Machine, see the state machine design diagram.

5.4.2 *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.

5.4.3 *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.

5.4.4 *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:

5.4.4.1 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 – see the Sandcastle documentation for an exhaustive description of all API elements – 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.

```
var myPipeline = PipelineController.Instance.PipelineIds[0].
```

6.1.1 *Scanning*

Scanning will find sensors and make them available for allocation and configuration within the API.

```
// Reference the sample programs for when these operations should occur.
myPipeline.Scan();
```

6.1.2 *Data Collection*

Data collection is handled via a delegate method. The following code will demonstrate the operation to initiate data collection, followed by a delegate method which gives access to the collected data.

```
// Set up the delegate for when data is collected and ready, the runtime of the data
// collection (120 seconds), and then begin data collection.
myPipeline.CollectionDataReady += DataReadyDelegate;
myPipeline.RunTime = Convert.ToDouble(120);
myPipeline.Start();
// . . .
```

```
// The delegate that fires whenever data is ready.

void DataReadyDelegate(object sender, ComponentDataReadyEventArgs e)
{
    // Prints out the number of channels of data we received.
    Console.WriteLine(e.Data.Length);
}
```

6.1.3 *Data Source Information Dictionary*

There are several properties you can extract 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.

```
var myDSInfo = myPipeline.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.

6.2 Sensors

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.)

```
var mySensor = PipelineController.Instance.PipelineIds[0].TrignoRfManager.Components[0];
```

6.2.1 *Sensor Type*

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

```
int sensorType = sensor.Properties.Type;
```

6.2.2 *Sensor Serial*

The unique Serial ID of the sensor.

```
int sensorSID = sensor.Properties.Sid;
```

6.2.3 *Sensor Firmware*

The firmware version of the sensor.

```
string sensorFW = sensor.Properties.Fw;
```

6.2.4 *Sensor Mode*

The configured mode of the sensor.

```
int sensorMode = sensor.Configuration.SampleMode;
```

6.2.5 *Sensor Channels*

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

```
var myChan = mySensor.TrignoChannels[0];
```

6.2.5.1 Sampling Rate

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

```
int sampleSize = myChan.SamplesPerFrame;
float frameInterval = myChan.FrameInterval;
// Equivalent to sampleSize/frameInterval
float sampleRate = myChan.SampleRate;
```

6.2.5.2 Units

The units of the values being output by the channel

```
var units = myChan.Units;
```

6.2.6 *Allocation/Deallocation*

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

```
var componentManager = myPipeline.TrignoRfManager;
// Allocates the sensor.
componentManager.SelectComponentAsync(mySensor);
// Deallocates the sensor.
componentManager.DeselectComponentAsync(mySensor);
```

6.3 References

6.3.1 *Sandcastle Documentation*

The Sandcastle Documentation included with your API package contains auto-generated descriptions of individual classes, methods, and properties. The below sources can get you started, but the Sandcastle Documentation will guide you to what the rest of what the API has to offer.

6.3.2 *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.

6.3.3 *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.