Delsys API

Quick Start Guide

MAN-032-1-2

Contents

1 Important Information

This document is meant only to provide an instructed implementation of the Delsys API. It is recommended you follow this guide and reference the Basic Streaming example included in the Delsys Example Application Github Repository should you need more context.

2 Quick Start

The following code snippets walk through the main aspects of setting up the Delsys API to perform a data collection. For more complete code, please see the relevant Delsys API sample application(s).

2.1 Initialize Data Source and Pipeline

The Data source 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.

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

_deviceSource = deviceSourceCreator.GetDataSource(SourceType.TRIGNO_RF);

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

After receiving the data source, the following commands allow us to set up a pipeline of RF data and access the Component and Transform Managers. Finally, we hook up to events allowing us to view:

```
Attempts to load device
try
{
    Create a Pipeline based on the datasource.
    PipelineController.Instance.AddPipeline(_deviceSource);
}
Catches exception if no base is detected
catch (BaseDetectionFailedException e)
{
    return;
}

Create a reference to this Pipeline for access to methods/properties
_pipeline = PipelineController.Instance.PipelineIds[0];

Register handlers for API component events
_pipeline.TrignoRfManager.ComponentAdded += ComponentAdded;
```

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

Register handlers for API collection events
_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.

2.2 Pair Sensor Components

Sensors must be paired to the base station or Trigno Lite before they are visible via the scan. You may specify the sensor number when pairing to a specific sensor. **This operation is optional if all of your sensors have already been paired to the system.**

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

2.3 Scan for Sensor Components

To detect components once the pipeline has been set up, you simply call the Scan function. This will find any sensors in range. **A scan must occur before you can stream your sensors.**

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

2.4 Configure Pipeline

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

```
bool[] sensorFlags = { };
DataLine dataLine = new DataLine(_pipeline, sensorFlags);
dataLine.ConfigurePipeline();


Get the frame throughput (the number of Trigno frames passed from the API at a time)
_frameThroughput = PipelineController.Instance.GetFrameThroughput();

// Get total number of channels
int totalChannels = 0;
foreach (var comp in _pipeline.TrignoRfManager.Components)
{
    totalChannels += comp.TrignoChannels.Count();
}

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

2.5 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.

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

Access collected data:

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

public void CollectionDataReady(object sender, ComponentDataReadyEventArgs e)
{
    //Channel based list of data for this frame interval
    List<List<double>> data = new List<List<double>>();

    for (int k = 0; k < e.Data.Count(); k++)
    {
        // Loops through each connected sensor
        for (int i = 0; i < e.Data[k].SensorData.Count(); i++)
        {
            // Loops through each channel for a sensor
            for (int j = 0; j < e.Data[k].SensorData[i].ChannelData.Count(); j++)
            {
                data.Add(e.Data[k].SensorData[i].ChannelData[j].Data);
            }
        }

    }

    //Add frame data to entire collection data buffer
    AllCollectionData.Add(data);
```

}

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();
```