Delsys API

Quick Start Guide

MAN-032-1-1

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 with your API package (examples/RF/Windows/Basic Streaming .NET) 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). For more information on Delsys API class and interface functionality, please see the Compiled HTML (chm) file located in the docs folder of your API package.

2.1 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.

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, we hook up to events allowing us to view:

```
//load the license files
var assembly = Assembly.GetExecutingAssembly();
string key;

using (Stream stream =
            assembly.GetManifestResourceStream("BasicExample.PublicKey.lic"))
{
    StreamReader sr = new StreamReader(stream);
    key = sr.ReadLine();
}

string lic;
using (Stream stream =
              assembly.GetManifestResourceStream("BasicExample.License.lic"))
{
    StreamReader sr = new StreamReader(stream);
    lic = sr.ReadToEnd();
}

//create the new portable device source
var devSource = new DeviceSourcePortable(key, lic);
//our device source is Trigno custom RF type.
var dev = devSource.GetDataSource(SourceType.TRIGNO_RF, Console.WriteLine);
//create a new pipeline for this device.
PipelineController.Instance.AddPipeline(dev);

//get the pipeline instance, and hook up some events.
var icManager = PipelineController.Instance.PipelineIds[0].TrignoRfManager;
icManager.ComponentScanComplete += ScanComplete;
icManager.ComponentAdded += SensorAdded;
icManager.ComponentRemoved += SensorRemoved;
icManager.ComponentPropChanged += SensorPropertyChanged;
```

ComponentAdded:

- We get an alert any time a component is added to the API subsystem.

PropertyChanged

- We get an alert anytime a component broadcasts an update in its properties.

2.2 Scan for Components

To detect components once the pipeline has been set up, you simply call the Scan function. This will find any sensors in range. This functionality works the same for both RF and Bluetooth.

```
bool result = await PipelineController.Instance.PipelineIds[0].Scan();
```

2.3 Configure Components and Data Source

The following code demonstrates how you can select or deselect paired sensors and add a new sensor to the list. You can also select from the available sample modes for each sensor.

```
//extract the available sample modes for this sensor, and apply.
string[] sm = sensor.Configuration.SampleModes;

//Apply the mode
var selector = new ModeSelector(sm);
selector.ShowDialog();
//Apply the desired sample mode.
sensor.Configuration.SelectSampleMode(selector.ReturnSampleMode());
samplemode_label.Text = sensor.Configuration.SampleMode.ToString();

//now select the sensor for data collection.
var success = await PipelineController.Instance.
              PipelineIds[0].TrignoRfManager.SelectComponentAsync(sensor);
```

Before collecting data, we configure the Datasource, and hook up to the Data Events. The data ready event will allow us to view collected data.

```
  TrignoDsConfig rfconfig = new TrignoDsConfig();
                rfconfig.ArmTrigger(false, false);

                //Add sensor configuration complete
                PipelineController.Instance.PipelineIds[0].CollectionDataReady +=
PCollectionDataReady;
                PipelineController.Instance.PipelineIds[0].CollectionComplete +=
PCollectionComplete;

                //apply the datasource configurations.
                bool result =
PipelineController.Instance.PipelineIds[0].ApplyInputConfigurations(rfconfig);

                if (!result)
                {
                    Console.WriteLine("Error");
                    return;
                }

                pipelinestate_label.Text =
PipelineController.Instance.PipelineIds[0].CurrentState.ToString();

                //apply the transform and sensor configurations.
                var outconfig = ConfigureTransforms();
                result =
PipelineController.Instance.PipelineIds[0].ApplyOutputConfigurations(outconfig);

                if (!result)
                {
                    Console.WriteLine("Error");
                    return;
                }
```

2.4 Configure Transforms and Arm

Finally, create and configure the desired transforms, after configuration is complete, arm the pipeline. This means that configurations have been applied and the pipeline is locked down for data collection. In order to change a configuration, the pipeline will need to be disarmed, configured, and re-armed.

```
//apply the transform and sensor configurations.
                var outconfig = ConfigureTransforms();
                result =
PipelineController.Instance.PipelineIds[0].ApplyOutputConfigurations(outconfig);
```

```
public OutputConfig ConfigureTransforms()
        {
            //we create a simple 1 to 1 mapping. One input channel will map to one output
channel.
            var tm = PipelineController.Instance.PipelineIds[0].TransformManager;
            var ic = PipelineController.Instance.PipelineIds[0].TrignoRfManager;

            //clear out the transforms.
            tm.TransformList.Clear();
            sensor_plot.Items.Clear();

            int numInChannel = 0;
            int numOutChannel = 0;

            //count the number of input channels we have.
            foreach (var tmp in ic.Components.Where(t => t.State ==
SelectionState.Allocated))
            {
                numInChannel += tmp.TrignoChannels.Count;
                numOutChannel += tmp.TrignoChannels.Count;
            }

            for (int i = 0; i < numInChannel; i++)
            {
                sensor_plot.Items.Add(i);
            }

            tm.AddTransform(new TransformRawData(numInChannel, numOutChannel));

            //channel configuration happens each time transforms are armed.
            var t0 = tm.TransformList[0];
            var outconfig = new OutputConfig();

            outconfig.NumChannels = numInChannel;
            int channelIndex = 0;

            foreach (SensorTrignoRf t in ic.Components)
            {
                //make sure sensor has been allocated for data collection
                if (t.State == SelectionState.Allocated)
                {
                    foreach (var chin in t.TrignoChannels)
                    {
                        //create a new output channel,add in and out chans, and map the
output.
                        var chout = new ChannelTransform(chin.FrameInterval,
chin.SamplesPerFrame, Units.VOLTS);
                        tm.AddInputChannel(t0, chin);
                        tm.AddOutputChannel(t0, chout);
                        outconfig.MapOutputChannel(channelIndex, chout); //which index do
we want to receive chout?
                        channelIndex++;
                    }
                }
            }

            return outconfig;
        }
```

2.5 Data Collection

Before starting data collection, set the collection time. Finally, issue a start command using the async/await pattern to wait for the collection result.

```
   //set the run time for the data collection.
            PipelineController.Instance.PipelineIds[0].RunTime =
int.Parse(textBox1.Text);

            //start the data collection.
            bool complete = await PipelineController.Instance.PipelineIds[0].Start();
```

Data will appear in the DataReady event created in section 2.3.

Access collected data:

```
        /// <summary>
        /// This is the main entry point for data collection.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PCollectionDataReady(object sender, ComponentDataReadyEventArgs e)
        {
            foreach (var point in e.Data[0].Data)
            {
                // This loop goes through each point collected by the first sensor
            }
        }
```

When data collection is complete, you can return to the disarm state using the following command.

```
var result = PipelineController.Instance.PipelineIds[0].DisarmPipeline();
```

For a more detailed list of API parameters and methods, please see the Sandcastle Documentation included with your API package.