Equivital Dongle SDK

To use the Dongle SDK you must first ensure that your project has references to both the SEMCoreDecoder.dll assembly (core SDK) and the EquivitalDongleExtension.dll assembly (dongle support).

You must also ensure the "SiUSBXp.dll" file is present in the same folder as the executable; we recommend adding this file to your project and setting its "Build Action" to "Content" and the "Copy to Output Directory" to "Copy if newer" to ensure this file is always deployed with the project's output assembly. In any event, the absence of this file will silently prevent the dongle working.

Interacting with the EDN:

The EDN is a windows service used by Equivital Applications to interact with the Equivital Dongle. It would be sensible to disable this so that the EDN does not steal the dongle connection from your own project.

The following code snippet can be used to turn off the EDN:

```
// Record whether the EDN was running so we can restart it.
bool ednWasRunning = false;

try
{
    if(ServiceController.GetServices().Count(x => x.ServiceName == "EDN Node Service") > 0)
    {
        ServiceController ednService = new ServiceController("EDN Node Service");
        ednWasRunning = ednService.Status == ServiceControllerStatus.Running;

        if (ednWasRunning)
        {
            Console.WriteLine("EDN Service running. Requesting it to stop");
            ednService.Stop();
            ednService.WaitForStatus(System.ServiceProcess.ServiceControllerStatus.Stopped);
        }
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

You should restart the EDN before terminating using the following code snippet:

```
if (ednWasRunning)
{
    try
    {
        Console.WriteLine("Restarting EDN service");
        ServiceController ednService = new ServiceController("EDN Node Service");
        ednService.Start();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Note: The ServiceController class is using System.ServiceProcess

If you do disable the EDN you should take steps to ensure it is always restarted otherwise Equivital Applications such as Equivital Manager and Equivital Professional will cease to function correctly.

Listening for SEM connections:

You can use the `EquivitalDongleManager` class to manage Equivital Dongle connections. This class is a singleton. You can access the instance using the static `Instance` method.

You should add an event handler for the "`ConnectionAdded`" and "`ConnectionRemoved`" events to handle raw SEM data connections from the Equivital Dongle.

When a new SEM is connected via the Equivital Dongle it will raise the "`ConnectionAdded`" event handler, and when a SEM is disconnected from the Equivital Dongle it will raise the "`ConnectionRemoved`" event handler. Generally just the `ConnectionAdded` event needs to be handled as the SDK contains events for disconnection.

The following code snippet sets up an event handler to handle new SEM connections:

```
EquivitalDongleManager.Instance.ConnectionAdded += DeviceConnectionAdded;
EquivitalDongleManager.Instance.Start();

.. rest of program ..

static void DeviceConnectionAdded(object sender, SemConnectionEventArgs e)
{
   // e.Connection contains the new ISemConnection instance.
   // Create a SemDevice instance and call Start(e.Connection) as per the SDK documentation.
}
```

Note: The "Start" method will start the manager running on a new thread so it will return immediately to the calling code. The event handler will almost certainly not be called on the UI thread so you may need to marshal UI interaction to the correct thread before interacting with user interface components.

Connecting to specific SEMs programmatically:

The `EquivitalDongleManager` will only attempt to connect to SEMs that it knows you are interested in. In order to tell the dongle manager which SEMs you want to connect to you need to call the "`AddBluetoothConnectableDevice`" method passing an instance of the `EquivitalBluetoothSensorInfo` class which describes each SEM you want to connect to.

To create an instance of the `EquivitalBluetoothSensorInfo` class you need to know the MAC address of the SEM and the security PIN code for it.

The following code snippet shows how to add a specific SEM to connect to:

```
var targetSensor = new EquivitalBluetoothSensorInfo("(00:01:02:03:04:05)", "1111");
EquivitalDongleManager.Instance.AddBluetoothConnectableDevice(targetSensor);
```

This code will get the Equivital Dongle Manager to try and connect to the SEM with MAC address of 00:01:02:03:04:05 and pin code 1111.

*Note that the factory default PIN code for all SEMs is "1111". If the specified PIN code is not correct then the dongle will fail (silently) to connect. You **must** ensure the code is correct.*

Discovering SEMs using Bluetooth Discovery:

The `EquivitalDongleManager` can automatically discover SEM devices in range. To do this you need to setup an event handler for the "`EquivitalSensorFound`" and "`SensorDiscoveryComplete`" events and call the "`DiscoverDevicesAsync`" method to start discovery.

The discovery process happens asynchronously so this method will return immediately to the caller. Each time a sensor has been found it will trigger the "`EquivitalSensorFound`" event handler. When discovery has concluded the "`DiscoverDevicesAsync`" event will be triggered. Once this event has been called no further discovery will take place and you can start the discovery process again if need be. *Please note that Bluetooth discovery can potentially cause disruption to data throughput for existing connections that may be running and is a resource intensive operation that should be used sparingly, and usually not during any important data capture.*

The "`EquivitalSensorFound`" event handler will be sent an instance of the `EquivitalBlueoothSensorInfo` class which can be used in conjunction with the "`AddBluetoothConnectableDevice`" method as described in the previous section. As with manually connecting a device you must remember to set the "`PinCode`" property.

The following code snippet demonstrates discovery:

```
EquivitalDongleManager.Instance.EquivitalSensorFound += DiscoveryEquivitalSensorFound;
EquivitalDongleManager.Instance.SensorDiscoveryComplete += DiscoverySensorDiscoveryComplete;
EquivitalDongleManager.Instance.DiscoverDevicesAsync();

.. rest of program ..



static void DiscoverySensorDiscoveryComplete(object sender, EventArgs e)
{
    // Discovery has now finished.
}

static void DiscoveryEquivitalSensorFound(object sender, EquivitalBluetoothInfoEventArgs e)
{
    // e.Info contains the EquivitalBluetoothSensorInfo instance for this SEM.
    // In this case we just add it to the manager.  We could store it or do something else.
    // We'll need to manually set the PIN code too; the SEM default being "1111".

    e.Info.PinCode = "1111";
    EquivitalDongleManager.Instance.AddBluetoothConnectableDevice(e.Info);
}
```

Note: The event handlers will be called asynchronously and not on the UI thread so any user interface interaction may need to be marshalled to the correct thread.

Terminating:

The `EquivitalDongleManager` should **always** be terminated before the application exits. This can be done very easily by calling the "`Terminate`" method.

The following code snipped shows how to terminate the manager:

```
EquivitalDongleManager.Instance.Terminate();
```

Note: Don't forget to unregister your event handlers from the manager before terminating!