Software development guide

For C# development

In this guide main classes are explained and usage of this API in software development is shown. Detailed class, method and event explanations can be found in the help file. Main classes explained here are TDTEngine, Session and Presentation. Basic implementation is explained with code samples. It is assumed that the developer using this guide is familiar with Visual Studio and can create mobile device solution and projects.

Main classes:

- ### TDTEngine

TDTEngine is a translation library which provides methods for translating electronic product code (EPC) between various levels of representation including BINARY, TAG_ENCODING, PURE_IDENTITY and LEGACY formats. It translates input data to the specified outbound level of the same coding scheme. The input data value may be a tag-encoding URI and the outbound level may be binary in which case the return value is a binary representation on that URI. This library contains definitions for EPC tags and EPC observations.

EPC observation is an object containing EPC code, date, RSSI and antenna properties. Data retrieved from the device is represented as collection of this object and is served through Presentation layer.

EPC code is an electronic product code and it is represented in EPCCode class. When this class is created with input objects of binary data from RFID tag or GTIN 14, serial number and company prefix types, data is translated into hexadecimal and binary data. It can be represented in binary, hexadecimal, tag encoding, pure identity URI, Legacy or ONS hostname level of representation. Conversion is done through EPC convert class which contains methods for translating EPC code from one level of representation to another.

- ### Session

Session layer is a threaded layer that deals with binary protocol communication over Bluetooth serial port, sending commands using communication queue, parsing received data and notifying through events. It is used to communicate with handheld reader and to provide simple function calls for device commands. It contains public methods exposed for usage if needed but it is mainly accessed through presentation layer. This layer uses interfaces for Communication Port, Communication Queue and Connection State.

For connecting and pairing of !D Hand and the phone without user interaction (automatically) BluetoothCommunicationPort class can be used. This class implements ICommunicationPort and uses 32feet.net library that provides support for Microsoft and Broadcom (Widcomm) Bluetooth stacks.

Other option is to use SerialCommunicationPort class which wraps SerialPort class and also implements ICommunicationPort interface. This implementation is Bluetooth stack independent, but requires a user to manually pair and connect phone to !D Hand Bluetooth Serial Port.

Communication Queue defines 2-way threaded communication queue which interacts with communication port and implements ICommunicationPort interface, so that BluetoothCommunicationPort or SerialCommunicationPort class can be used as port implementation.

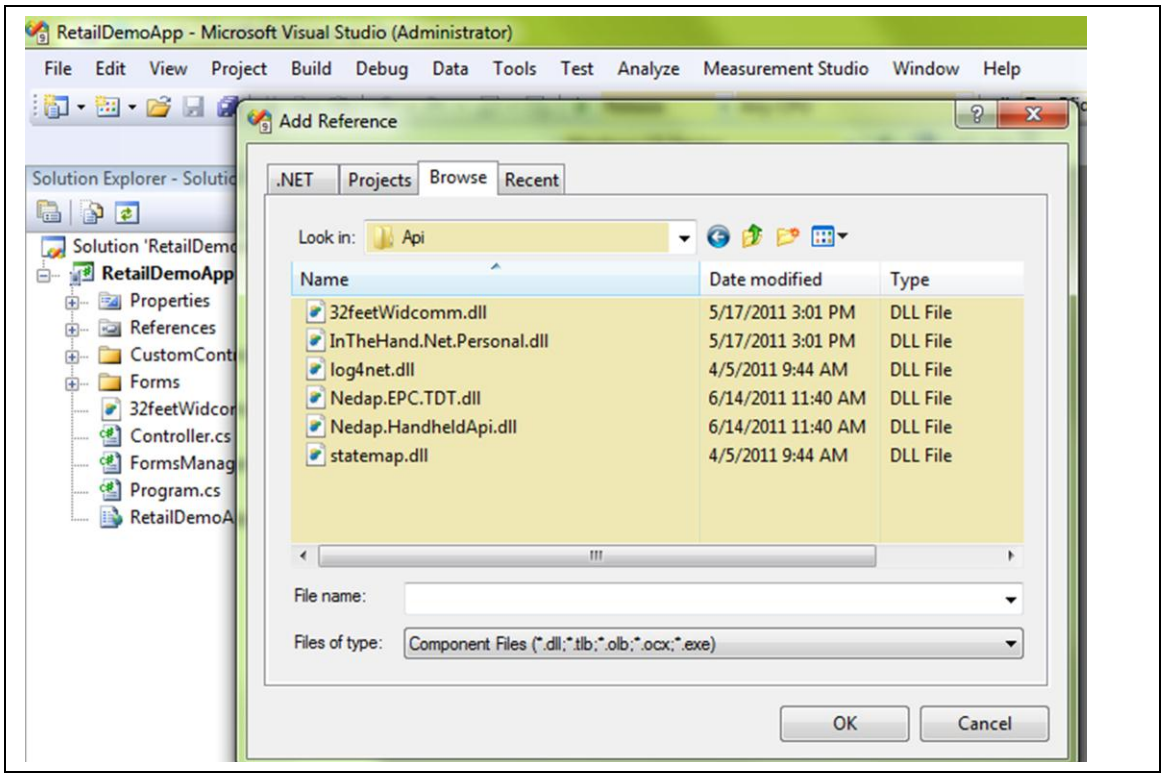
ConnectionState class is used by the Session for connect/disconnect detection. It implements IConnectionState interface, and runs in separate thread that periodically sends/receives a message.

- ### Presentation

Presentation is a high-level communication layer used to communicate with a handheld reader. It is built on top of the session layer and provides more sophisticated functionality. This layer functions are calling functions from session layer. Once the answer is obtained from the device, presentation layer is notified via event. This layer contains public events and methods for all implemented functionality and it is accessed from GUI layer of application. Use of presentation layer is recommended for interaction with handheld reader.

Beginning development:

Developing application based on this API requires few steps in order to get full functionality. First you need to add libraries to you project:


RetailDemoApp - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Analyze Measurement Studio Window Help
Solution Explorer - Solutic
Solution 'RetailDemo
RetailDemoApp
Properties
References
CustomConti
Forms
32feetWidco
Controller.cs
FormsManag
Program.cs
RetailDemoA
Add Reference
.NET
Projects
Browse
Recent
OK
Cancel

Look in: Api

Name	Date modified	Type
32feetWidcomm.dll	5/17/2011 3:01 PM	DLL File
InTheHand.Net.Personal.dll	5/17/2011 3:01 PM	DLL File
log4net.dll	4/5/2011 9:44 AM	DLL File
Nedap.EPC.TDT.dll	6/14/2011 11:40 AM	DLL File
Nedap.HandheldApi.dll	6/14/2011 11:40 AM	DLL File
statemap.dll	4/5/2011 9:44 AM	DLL File

File name:

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe)

Libraries that should be added are: InTheHand.Net.Personal.dll, log4net.dll, Nedap.EPC.TDT.dll, Nedap.HandheldApi.dll, statemap.dll. InTheHand.Net.Personal.dll library requires native library 32feetWidcom.dll for Widcomm (Broadcom) Bluetooth stack support, and this library should be added as a file to project and in properties of this file 'Copy to Output Directory' should be set to 'Copy always', and 'Build Action' to 'Content'.

Next thing is to create TDTEngine object and Initialize it. It can be initialized without parameters or with parameters of XMLDefinitions type which is public enumerator in TDTEngine. This list represents translation definitions XML files.

For usage of all available definitions Initialize can be called without parameters:

```
…
Nedap.EPC.TDT.TDTEngine engine = new Nedap.EPC.TDT.TDTEngine();
engine.Initialize();
…
```

For usage of specific definitions Initialize with these XmlDefinitions can be called:

```
…
Nedap.EPC.TDT.TDTEngine engine = new Nedap.EPC.TDT.TDTEngine();
XmlDefinitions[] definitionsList = { XmlDefinitions.SGLN_96,
XmlDefinitions.SGTIN_96 };
engine.Initialize(definitionsList);
…
```

Next step is to create ICommunicationPort, Session and PresentationFacade.

```
…
public ICommunicationPort Port { get; private set; }
public Session Session { get; private set; }
public PresentationFacade Facade { get; private set; }
…
```

Here are two options:

1.) If the phone uses Microsoft or Widcomm (Broadcom) Bluetooth stack (32feet.net library), device discovery, pairing and connecting (implemented in BluetoothCommunicationPort) can be done without user interaction:

```
…
Port = new BluetoothCommunicationPort();
Session = new Session(Port, true);
Facade = new PresentationFacade(Session);
…
```

2.) If the phone does not support mentioned Bluetooth stacks, SerialCommunicationPort class can be used. This approach requires the user to use target phone Bluetooth software to manually discover, pair and connect to the !D Hand Bluetooth Serial Port. Port name (portName) is the name of the System.IO.Ports.SerialPort that is used to connect to the handheld reader (e.g.

"COM1"). Note that port name can be provided from UI and that user should select COM port used to connect to the !D Hand.

```
…
Port = new SerialCommunicationPort(portName, 9600, Parity.None, 8,
StopBits.One);
Session = new Session(Port, true);
Facade = new PresentationFacade(Session);
…
```

A universal solution would be to try to instantiate BluetoothCommunicationPort, and if platform is not supported an exception would be thrown. This exception can be catch, and fallback to SerialPort class can be done. With this approach more robust solution can be created.

After these steps interaction with !D Hand can be done through PresentationFacade with method calls and event handling. For example, for reading Rfid tags these calls should be made:

1.) Subscribe to event, and enter RfidInventoryStart state (!D Hand is ready for reading)

```
…
Program.Controller.Facade.OnReadRfidTag += new
System.EventHandler<ReadRfidTagEventArgs>(facade_OnReadRfidTag);
Program.Controller.Facade.RfidInventoryStart();
…
```

2.) event handler, and delegate for UI update (note that BeginInvoke should be used since this event is triggered from another thread)

```
…
private void facade_OnReadRfidTag(object sender, ReadRfidTagEventArgs e)
{
      this.BeginInvoke(new UpdateUIDelegate(FormUpdateMethod),
e.EpcObservation);
}

private void FormUpdateMethod(EpcObservation epcObservation)
{
      lbInventoryItemsList.Items.Add(epcObservation.EpcCode.HexString);
}
…
```

When reading is finished, the code should unsubscribe from event stop inventory call should be made:

```
…
Program.Controller.Facade.OnReadRfidTag -= new
System.EventHandler<ReadRfidTagEventArgs>(facade_OnReadRfidTag);
Program.Controller.Facade.HandheldStop();
…
```

On applicaiotn exit, a dispose of classes used for interaction should be done in this order:

```
public void Dispose()
{
      if (Session != null)
      {
            Facade.Dispose();
            Port.Dispose();
            Session.Dispose();
      }
}
```

This sums up basic information and code examples for beginning development with this API. More details can be found in an example application (RetailDemoApp).