How this Script Works:

1) The Start() function is recognised by Unity as the position to start the script and is called once at the start. Firstly, it searches for the 3 GameObjects – the panels, and shows the scanning panel first. The BLE module is initialised at line 302 with the initialise() function.
2) The Invoke("Scan"), then runs the Scan() function after 1 second delay.

```
308     void Start()
309     {
310
311         panelScan = GameObject.Find("panelScan");
312         panelSettings = GameObject.Find("panelSettings");
313         panelConnected = GameObject.Find("panelConnected");
314
315         // set up the panels
316         showScan();
317
318         // initialise the bluetooth library
319         Initialise();
320
321         // start scanning after 1 second
322         Invoke("scan", 1000);
323     }
```

```
302 void Initialise()
303 {
304     BluetoothLEHardwareInterface.Initialize(true, false, () => { }, (error) => { });
305 }
```

3) The scan function runs (line 154), and will remove any old peripherals, then add new peripherals:

```
154 public void scan()
155 {
156     if (_scanning == true)
157     {
158         txtDebug.text += "Stop scan\n";
159         BluetoothLEHardwareInterface.StopScan();
160         _scanning = false;
161     }
162     else
163     {
164
165         txtDebug.text += "Start scan\n";
166         RemovePeripherals();
167
168         devicesFound = 0;
169
170         // the first callback will only get called the first time this device is seen
171         // this is because it gets added to a list in the BluetoothDeviceScript
172         // after that only the second callback will get called and only if there is
173         // advertising data available
174         BluetoothLEHardwareInterface.ScanForPeripheralsWithServices(null, (address, name) => {
175             AddPeripheral(name, address);
176         }, (address, name, rssi, advertisingInfo) => { });
177
178         _scanning = true;
179     }
180 }
```

4) The addPeripheral() function creates a new button game object in the scroll panel, which has a behaviour that fills in the global variables: TextName = The name of the BLE device, and TextAddress = the device address. These are variables found within the addPeripheral() class when scanning.

```
197     void AddPeripheral(string name, string address)
198     {
199         if (_peripheralList == null)
200         {
201             _peripheralList = new Dictionary<string, string>();
202         }
203         if (!_peripheralList.ContainsKey(address))
204         {
205
206             txtDebug.text += "Found " + name + "\n";
207             devicesFound++;
208
209             GameObject buttonObject = (GameObject)Instantiate(connectButton);
210             connectButtonScript script = buttonObject.GetComponent<connectButtonScript>();
211             script.TextName.text = name;
212             script.TextAddress.text = address;
213             script.controllerScript = this;
214
215             // each button is 50 pixels high
216             // the container panel is 544 pixels high
217             var h = (980 / 2) - (55 * devicesFound);
218
219             buttonObject.transform.SetParent(PanelScrollContents);
220             buttonObject.transform.localScale = new Vector3(1f, 1f, 1f);
221             buttonObject.transform.localPosition = new Vector3(0, h, 0);
222
223             _peripheralList[address] = name;
224
225             txtDebug.text += "Button created\n";
226         }
227     }
```

5) Once a button has been created, if it is clicked (the function is within the connectButtonScript.cs), the connectTo() function is run. This stops the BLE plugin from scanning, then runs the connectBluetooth() function.

```
138     public void connectTo(string sName, string sAddress)
139     {
140         if (_connecting == false)
141         {
142             txtDebug.text += "Connect to " + sName + " " + sAddress + "\n";
143             _connecting = true;
144
145             // stop scanning
146             BluetoothLEHardwareInterface.StopScan();
147             _scanning = false;
148
149             // connect to selected device
150             connectBluetooth(sAddress);
151         }
152     }
```

6) Using the connectBluetooth() function, the BLE plugin will connect to the peripheral, if the characteristics are equal to those specified at the start of the script. First it will check the serviceUUID, and if equal, set the connected state to True. Then the two sub checks are the read and write abilities. If these also match, then both abilities will occur. If the serviceUUID does not match, the device will not connect, and nothing will happen at the button click.

```
81      void connectBluetooth(string addr)
82      {
83          BluetoothLEHardwareInterface.ConnectToPeripheral(addr, (address) => {
84          },
85          (address, serviceUUID) => {
86          },
87          (address, serviceUUID, characteristicUUID) => {
88
89              // discovered characteristic
90              if (IsEqual(serviceUUID, _serviceUUID))
91                  {
92                      _connectedID = address;
93                      isConnected = true;
94
95                      if (IsEqual(characteristicUUID, _readCharacteristicUUID))
96                      {
97                          _readFound = true;
98                      }
99                      else if (IsEqual(characteristicUUID, _writeCharacteristicUUID))
100                     {
101                         _writeFound = true;
102                     }
103
104                     showConnected();
105                 }
106         }, (address) => {
107
108             // this will get called when the device disconnects
109             // be aware that this will also get called when the disconnect
110             // is called above. both methods get call for the same action
111             // this is for backwards compatibility
112             isConnected = false;
113         });
114
115         _connecting = false;
116     }
```

7) Upon successful connection, the showConnected() function will run, which sets the connected panel visibility to On. With this, we can subscribe to the data coming in and out from the device. The readfound and writefound parameters must both be true for this main Update() function to work. These are set in instruction 6). The main Update() function will now listen for data and display it when it sees it. I am unsure yet if the send data button function will work, although I doubt it.