App Inventor, Micro:bit and UART

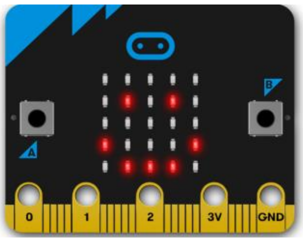
November, 2021

Somehow, communicating with a micro:bit via UART is more difficult than it should be. For now, using the extensions as indicated in this tutorial, will ensure that everything works.

This tutorial is intended to show how to use Bluetooth Low Energy (BLE) to send messages to a micro:bit. This can be used to control a toy robot car or any device that uses a micro:bit as its brain.

This first part will only cover the basic communication with a micro:bit via BLE. A follow up tutorial will cover controlling a robot car, for example the Ring:bit car v2 or the DFRobot Maqueen, both available for less than 30$, not including a micro:bit, which should cost less than 20$ for a v2 version.

The micro:bit is an electronics board, intended for schoolchildren, to learn programming and simple electronics.



Stand-alone, you can program a micro:bit to display text, little figures on the small LED screen and it supports some functions like temperature measurement and an accelerometer. It can be programmed using Microsoft MakeCode (a block language), JavaScript, Python and Scratch.

For more information, see https://microbit.org/

The micro:bit supports Bluetooth Low Energy (BLE) and therefore it can communicate with an App Inventor app, using the App Inventor BLE extension. There is also a micro:bit extension that can be used together with the BLE extension.

Using these extensions you can send text to the micro:bit, you can put smileys and the like on the little LED-screen, you can read temperature values, the magnetometer and accelerometer values from the micro:bit and display them on your phone.

Micro:bits start to become more interesting when they are used as the brain of a toy robot car, robot arm, and so on. And it becomes even more interesting if we could control these devices with our phones, using App Inventor. This can be done by sending commands to a micro:bit, which translates these to commands for the robot car. The only way to send such messages is via UART (Universal Asynchronous Receiver/Transmitter), a serial messaging protocol using BLE.

There are several problems though. First of all, at the time of this writing, there is no UART part for the latest micro:bit extension. Secondly, the memory of the v1 micro:bit is so small, that the memory hungry Bluetooth support is very restricted. Fortunately the memory of the v2 micro:bit is much larger: 16kB RAM for the v1 version and 128kB RAM for v2. The Flash memory has gone from 256kB to 512kB.

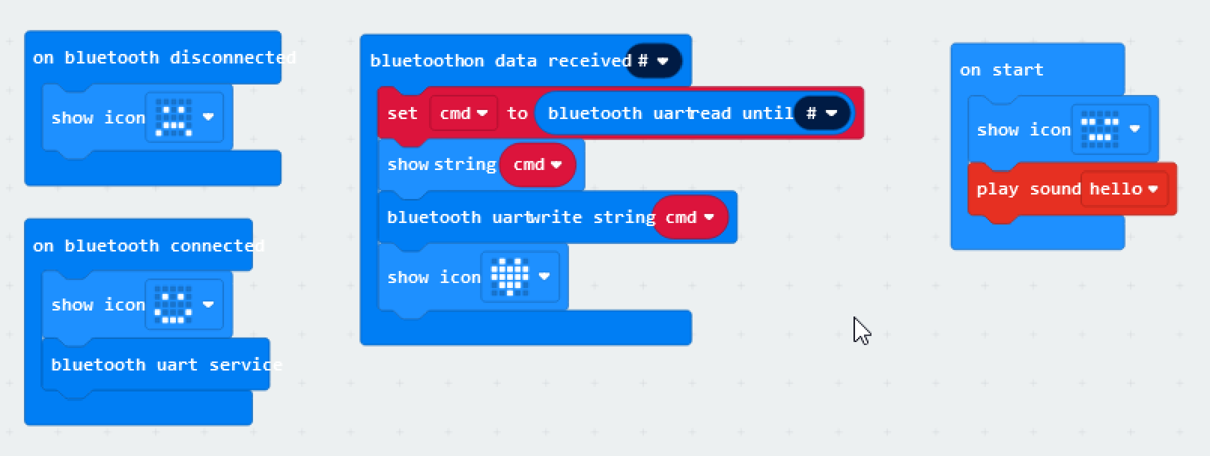
Preparing the micro:bit

To be able to experiment with App Inventor code, we need to prepare a micro:bit first. This is not a tutorial about micro:bit coding, but if you know some basic App Inventor, the blocks in MakeCode are rather straightforward.

When you start a new project, you will not see any blocks that can be used with Bluetooth. Bluetooth is an extension that needs to be loaded first:

- Click **Advanced** and then + Extensions
- You will see a screen with extensions. Search for Bleutooth
- A block with Bluetooth services will appear. Click on it.
- A pop-up will give a warning that the radio extension is incompatible and will be removed.
- Click Remove Extension and add Bluetooth.

Now, create a set of blocks looking like this:



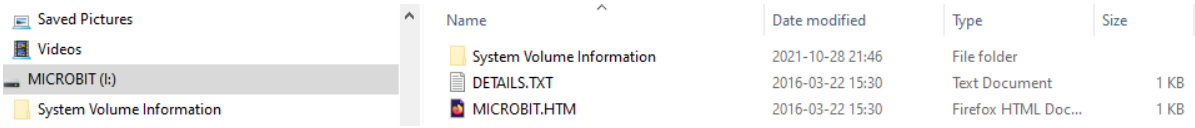
```
on bluetooth disconnected
    show icon

on bluetooth connected
    show icon
    bluetooth uart service

bluetoothon data received #
    set cmd to bluetooth uartread until #
    show string cmd
    bluetooth uartwrite string cmd
    show icon

on start
    show icon
    play sound hello
```

- *On start* shows an icon and a sound is palyed. Playing sounds is only possible with a micro:bit v2, which is strongly advised to use, because of the larger memory.
- When Bluetooth is connected, start the UART service. This block can be found in the …more part of the Bluetooth blocks.
- An event handler, *Bluetooth data received* allows us to read the data that is coming from our App Inventor app. This data should be delimited by some character that you do not intend to use otherwise, in the message itself. We choose the **#** character.
- The uartread block is also in the …more section. As is the uartwrite block.
- These blocks just receive the data and echo back what it received.
- Then show a hart, indicating the sending of data is finished.

Now, if you want to save yourself a lot of headache, click on the little wheel at the right top and then click “Project Settings”. You will see a list of pairing options. Choose “**No Pairing Required**”. Unless you are sitting in a class room or in a public space, this is really the best option.

The next step is flashing this code onto your micro:bit. Click the *Save* or *Download* button. In both cases a .hex file will be downloaded to your download folder with the name you specified in the input field next to the Save button.

Warning: If you are using a Chrome based browser, the browser will try to convince you to pair your micro:bit to your PC and flash your micro:bit via Bluetooth. *Do not do that!* It will upset the Bluetooth settings on your micro:bit. *Instead*, attach the micro:bit via a good old micro-USB cable to your PC. The PC will recognize a new drive and in your file explorer you will see a picture like this:



Name	Date modified	Type	Size
System Volume Information	2021-10-28 21:46	File folder	
DETAILS.TXT	2016-03-22 15:30	Text Document	1 KB
MICROBIT.HTM	2016-03-22 15:30	Firefox HTML Doc...	1 KB

Locate the .hex file you just downloaded in your download folder and drag that file onto the micro:bit drive folder. In a few moments you will see green bubbles appearing in a pop-up window and when it indicates 100%, your micro:bit is ready for use. You will see the icon you coded in the *on start* block and you will hear the sound.

For our experiment you could leave the micro:bit attached to the PC with the USB cable, because it will provide power to the micro:bit. Otherwise, attach a battery box with two AAA batteries in it.

Coding the App Inventor App

First we have to set up a basic app that can connect to a BLE device. Head over to:
http://iot.appinventor.mit.edu/assets/howtos/MIT_App_Inventor_Basic_Connection.pdf

But: download the BLE extension from here: (instead of the link given in the PDF, which is old)
http://iot.appinventor.mit.edu/assets/resources/edu.mit.appinventor.ble-20200828.aix

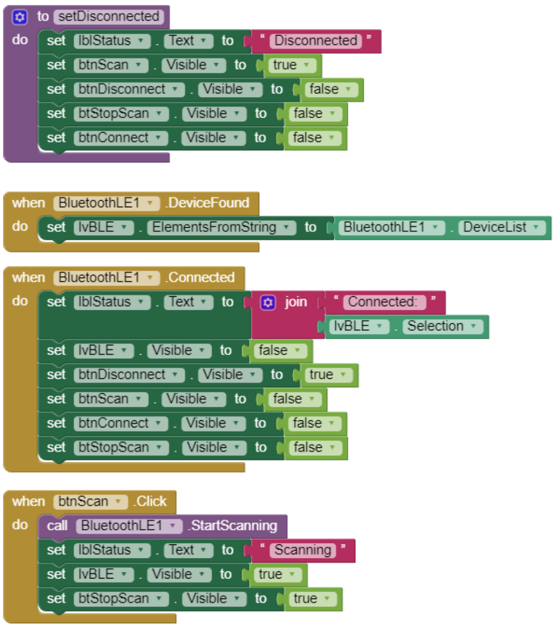
At the time of writing, this is the latest one. If there is a later one, you probably find it here:
http://iot.appinventor.mit.edu/#/bluetoothle/bluetoothleintro

Do NOT download the micro:bit extension found here:
http://iot.appinventor.mit.edu/#/microbit/microbitintro
It does not have UART support.

Instead, download the oldest micro:bit extension, found here:
http://iot.appinventor.mit.edu/assets/resources/com.bbc.microbit.profile.aix

Build an app as described in the MIT App Inventor Basic Connection pdf, **using the extensions given here** instead of the extensions described in the pdf.

You can also use the starter app provided here. The functionality is the same, the looks are slightly different. The blocks we used, look like this:

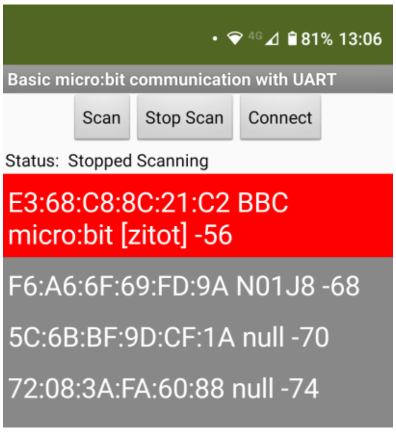


```
to setDisconnected
do  set lblStatus . Text to "Disconnected"
    set btnScan . Visible to true
    set btnDisconnect . Visible to false
    set btStopScan . Visible to false
    set btnConnect . Visible to false

when BluetoothLE1 .DeviceFound
do  set lvBLE . ElementsFromString to BluetoothLE1 . DeviceList

when BluetoothLE1 .Connected
do  set lblStatus . Text to join "Connected: "
                                 lvBLE . Selection
    set lvBLE . Visible to false
    set btnDisconnect . Visible to true
    set btnScan . Visible to false
    set btnConnect . Visible to false
    set btStopScan . Visible to false

when btnScan .Click
do  call BluetoothLE1 .StartScanning
    set lblStatus . Text to "Scanning"
    set lvBLE . Visible to true
    set btStopScan . Visible to true

when btStopScan .Click
do  call BluetoothLE1 .StopScanning
    set lblStatus . Text to "Stopped Scanning"
    set btnConnect . Visible to true

when Screen1 .Initialize
do  call setDisconnected

when BluetoothLE1 .Disconnected
do  call setDisconnected

when btnConnect .Click
do  call BluetoothLE1 .StopScanning
    call BluetoothLE1 .Connect
                        index lvBLE . SelectionIndex
    set lblStatus . Text to "Connecting"

when btnDisconnect .Click
do  call BluetoothLE1 .Disconnect
```

There is a slight difference of our starter app with the app described in the pdf. We tried to make the user interface slightly better looking, but the functionality is basically the same.

Connecting to the micro:bit

- Power on the micro:bit, either by using the USB cable or by using a battery box.
- When you started the app, by connecting it to the AI Companion, you will see an empty screen with just the Scan button.
- Clicking the button will show a list of near-by Bluetooth devices, the Stop Scanning Button and the Connect button..
- If you see a device with BBC micro:bit in the name, click Stop Scanning.
- Click on the name of your micro:bit and click connect.
- You should see a smiley on the micro:bit and a message on your phone.



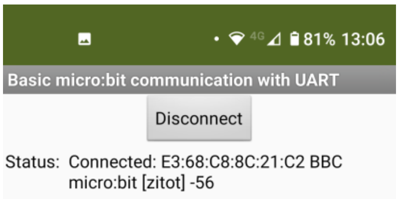
Status: Stopped Scanning

E3:68:C8:8C:21:C2 BBC micro:bit [zitot] -56

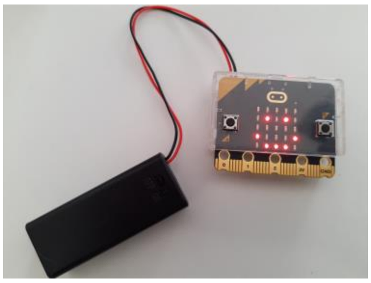
F6:A6:6F:69:FD:9A N01J8 -68

5C:6B:BF:9D:CF:1A null -70

72:08:3A:FA:60:88 null -74



Status: Connected: E3:68:C8:8C:21:C2 BBC micro:bit [zitot] -56



Communicating with the micro:bit and Adding UART Support

For more information about the micro:bit and Bluetooth, look here: https://tech.microbit.org/bluetooth/ and here: https://lancaster-university.github.io/microbit-docs/ble/profile/#introduction

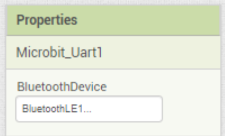
There is a lot of information about Bluetooth itself on the web, but a straightforward description of how to use it in your app's are hard to find. But, as said, we only need some basic UART facilities to send messages back and forth, which is covered by the extensions available.

Our starting point for this part of the tutorial is the app we just built that allows us to connect to a micro:bit. Remember that our micro:bit is already set up for the communication we need.

Instead of building the app as shown below, you could look at the completed app.

First, save the project you have so far on your PC by going to projects->Export selected project. Then, under the same tab, save your project with a different name. This is because you want to preserve what you have done so far.

If you did not do it yet, download the micro:bit extension from here:
http://iot.appinventor.mit.edu/assets/resources/com.bbc.microbit.profile.aix



Properties

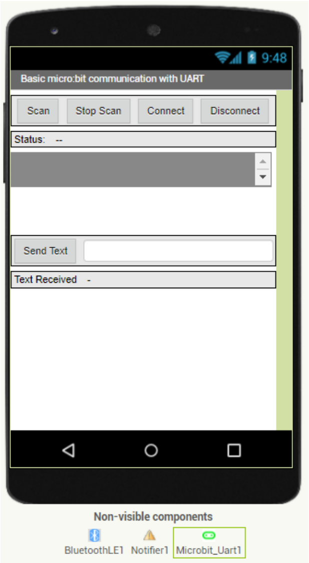
Microbit_Uart1

BluetoothDevice

BluetoothLE1...

Import the extension you just downloaded. Drag *Microbit_Uart* to your design. Click on it, and in its properties window, indicate that *BluetoothLE1* is its associated Bluetooth device.

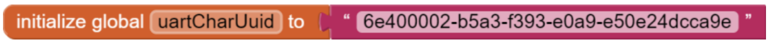
Adapt the design to include a horizontal layout with a button and a text field. The text field will contain the text that is sent to the micro:bit when the button is pressed. Add another horizontal layout with two labels, one saying "Text Received" and the other will contain the received text later.



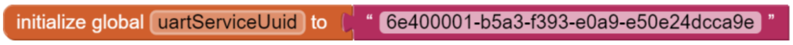
Also add a WebViewer and make it not visible. We will need it later to execute some JavaScript under the hood.

The Blocks

Go to the **blocks tab**. Define two global variables:

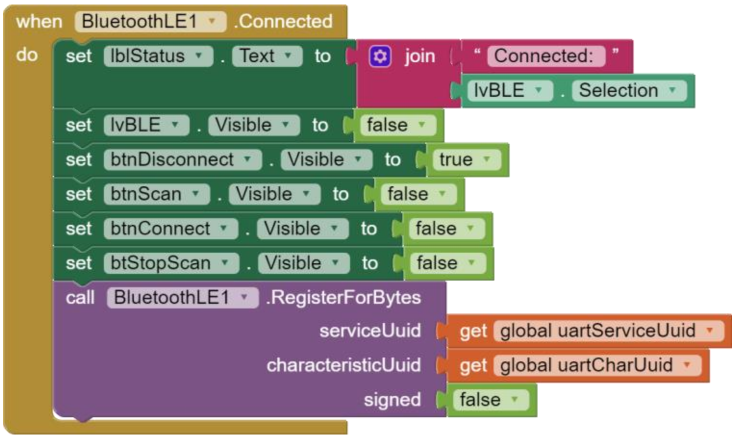


```
initialize global uartCharUuid to "6e400002-b5a3-f393-e0a9-e50e24dcca9e"
```



```
initialize global uartServiceUuid to "6e400001-b5a3-f393-e0a9-e50e24dcca9e"
```

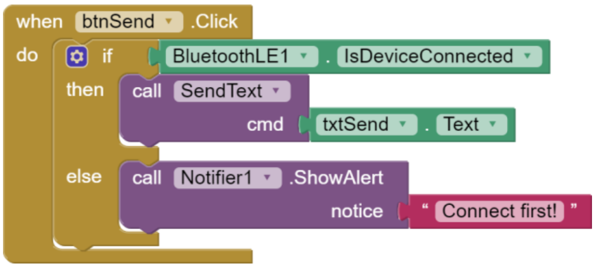
Then, adapt the *when.BluetoothLE1.Connected* blocks to look like this:



```
when BluetoothLE1 .Connected
do  set lblStatus . Text to  join  "Connected: "
                                   lvBLE . Selection
    set lvBLE . Visible to false
    set btnDisconnect . Visible to true
    set btnScan . Visible to false
    set btnConnect . Visible to false
    set btStopScan . Visible to false
    call BluetoothLE1 .RegisterForBytes
                  serviceUuid  get global uartServiceUuid
           characteristicUuid  get global uartCharUuid
                       signed  false
```

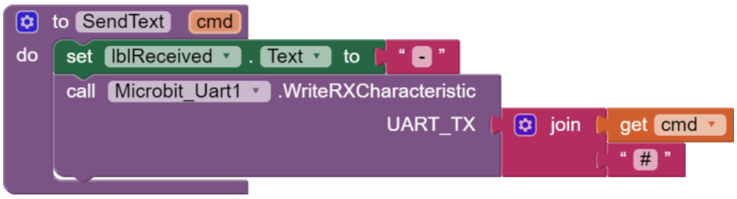
What is needed, is the *RegisterForBytes*, this ensures that the micro:bit will send acknowledgements to your App Inventor app.

The next block we will add is the block that sends a text message to the micro:bit.



```
when btnSend .Click
do  if    BluetoothLE1 . IsDeviceConnected
    then  call SendText
               cmd  txtSend . Text
    else  call Notifier1 .ShowAlert
               notice  "Connect first!"
```

We defined a function to do the actual sending:

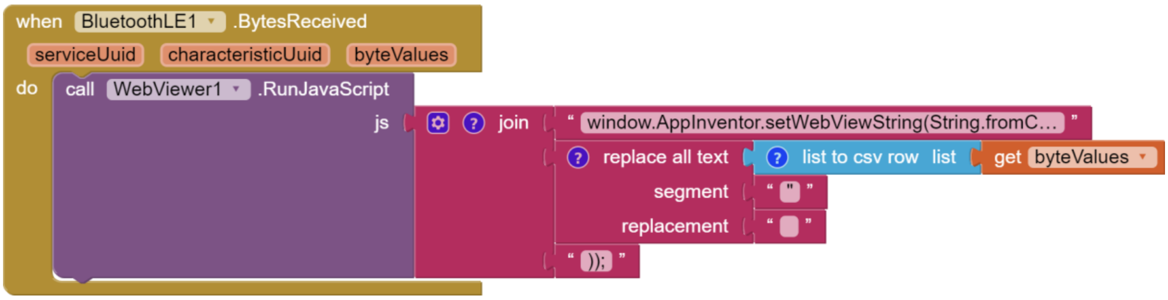


```
to SendText cmd
do  set lblReceived . Text to " "
    call Microbit_Uart1 .WriteRXCharacteristic
                 UART_TX  join  get cmd
                                "#"
```

Note that we join the text given with a # character, This character is also used in the micro:bit code to delimit a message. This character should not be used for other purposes, or your logic for your app may fail.

We use here the only block needed from the micro:bit extension. Theoretically we would not need it, but we would need to convert our string to a list of bytes and we would need to know which UUID's to use.

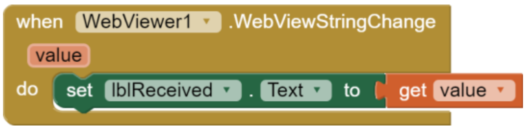
Unfortunately, that is what we receive from the micro:bit: a list of bytes. The block which should convert it for us, does not work properly, therefore we resort to a JavaScript trick, using the hidden WebViewer we added earlier.



```
when BluetoothLE1 .BytesReceived
  serviceUuid  characteristicUuid  byteValues
do  call WebViewer1 .RunJavaScript
                            js  join  " window.AppInventor.setWebViewString(String.fromC… "
                                      replace all text  list to csv row  list  get byteValues
                                               segment  " " "
                                           replacement  "  "
                                      " )); "
```

The string in the join block reads like this:
Window.AppInventor.setWebViewString(String.fromCharCode(

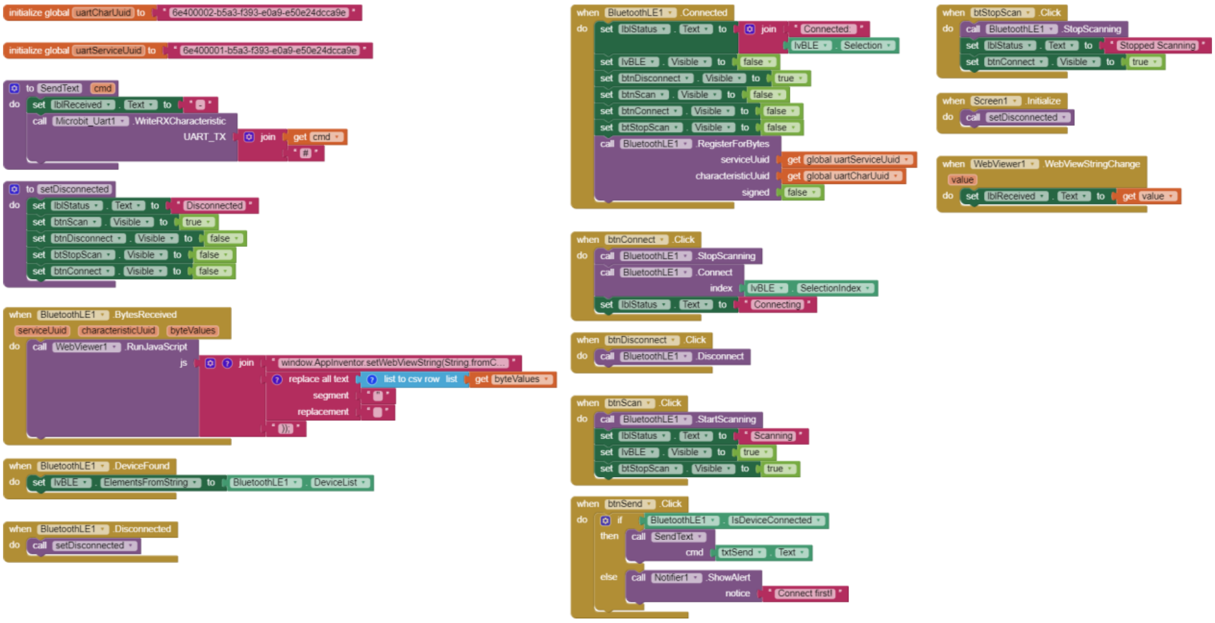
The function will convert the ASCII-bytes returned by the micro:bit, to a string.



```
when WebViewer1 .WebViewStringChange
  value
do  set lblReceived . Text to  get value
```

The *WebViewStringChange* event will return the converted string that we can display in our label.

Here is the complete set of blocks:



```
initialize global uartCharUuid to  " 6e400002-b5a3-f393-e0a9-e50e24dcca9e "

initialize global uartServiceUuid to  " 6e400001-b5a3-f393-e0a9-e50e24dcca9e "

to SendText  cmd
do  set lblReceived . Text to  " - "
    call Microbit_Uart1 .WriteRXCharacteristic
                          UART_TX  join  get cmd
                                         " # "

to setDisconnected
do  set lblStatus . Text to  " Disconnected "
    set btnScan . Visible to  true
    set btnDisconnect . Visible to  false
    set btStopScan . Visible to  false
    set btnConnect . Visible to  false

when BluetoothLE1 .BytesReceived
  serviceUuid  characteristicUuid  byteValues
do  call WebViewer1 .RunJavaScript
                            js  join  " window.AppInventor.setWebViewString(String.fromC… "
                                      replace all text  list to csv row  list  get byteValues
                                               segment  " " "
                                           replacement  "  "
                                      " )); "

when BluetoothLE1 .DeviceFound
do  set lvBLE . ElementsFromString to  BluetoothLE1 . DeviceList

when BluetoothLE1 .Disconnected
do  call setDisconnected

when BluetoothLE1 .Connected
do  set lblStatus . Text to  join  " Connected: "
                                   lvBLE . Selection
    set lvBLE . Visible to  false
    set btnDisconnect . Visible to  true
    set btnScan . Visible to  false
    set btnConnect . Visible to  false
    set btStopScan . Visible to  false
    call BluetoothLE1 .RegisterForBytes
                      serviceUuid  get global uartServiceUuid
               characteristicUuid  get global uartCharUuid
                           signed  false

when btnConnect .Click
do  call BluetoothLE1 .StopScanning
    call BluetoothLE1 .Connect
                      index  lvBLE . SelectionIndex
    set lblStatus . Text to  " Connecting "

when btnDisconnect .Click
do  call BluetoothLE1 .Disconnect

when btnScan .Click
do  call BluetoothLE1 .StartScanning
    set lblStatus . Text to  " Scanning "
    set lvBLE . Visible to  true
    set btStopScan . Visible to  true

when btnSend .Click
do  if  BluetoothLE1 . IsDeviceConnected
    then  call SendText
                  cmd  txtSend . Text
    else  call Notifier1 .ShowAlert
                  notice  " Connect first! "

when btStopScan .Click
do  call BluetoothLE1 .StopScanning
    set lblStatus . Text to  " Stopped Scanning "
    set btnConnect . Visible to  true

when Screen1 .Initialize
do  call setDisconnected

when WebViewer1 .WebViewStringChange
  value
do  set lblReceived . Text to  get value
```

Try it out! Connect your app to your phone with the Companion, power-up the micro:bit, connect to it and start sending some text, like "Hello microbit".

In the next tutorial we will control a little robot car. This is quite easy to do, just send specific commands to the micro:bit, that it will use to steer the car. There are some timing constraints though, that have to be taken care of.