Creating iPhone and iPad Apps Made Easy

Control external devices using WiFi or BLE

By **Hans Oppermann** (Germany)

There are several obstacles to be overcome before you can start writing applications for iPhones and iPads, and there are many cases where hobbyists who are just looking for a rudimentary app to control some external hardware would welcome a simple solution. Enter techBASIC: this is a platform that enables you to write your own iPhone and iPad applications directly on your mobile device, and it comes with a suite of libraries covering a wide range of functions. In this article we look at an example application that communicates with the outside world using BLE (Bluetooth Low Energy).

Apple provides a free development environment called Xcode for developing your own apps. Normally the Swift programming language is used, and the current version, which is version 8, requires a Mac computer with an up-to-date operating system. Once the app has been built it can be transferred directly from Xcode to the mobile device.

The techBASIC app

Even if all you want to do is use your smartphone to control an external device and provide a simple and straightforward user interface (plus the specific code necessary to drive your circuit), using Xcode is a lot of bother. An interesting alternative is the 'techBASIC' app by Byte Works [1].
The entire integrated development environment (IDE) is contained within its own app, which is available as a demo version (called techSampler), and as a full-featured version for around US$20, downloadable from the app store [2]. The demo version only allows built-in demo programs to be executed: is is not possible to write your own programs or modify existing ones.
When the app is launched all the available programs (or demos) are displayed in a directory. It is possible to run or edit programs from this listing. It is also possible to convert a finished program into a standalone app, which requires the techBASIC App Builder program (priced at US$49) and a Mac computer on which the app is built. However, for most hobbyists this is an unnecessary overcomplication.
Developing programs on an iPhone is far from ideal because of its small display. An iPad with an external keyboard makes a much better platform for writing code, whether the target device is an iPad or an iPhone. It is of course only necessary to purchase the techBASIC app once.

Listing 1. Processing user input: example of a WiFi client communicating with an ESP8266.

```
! Handle pressed button
SUB touchUpInside (ctrl AS Button, time AS DOUBLE)
   http_string = "http://192.168.254.70/"

! Heizung EIN
    IF ctrl = Tag THEN
        http_string = http_string & "gpio/1"
        proc_http
    END IF
! Heizung AUS
    IF ctrl = Nacht THEN
        http_string = http_string & "gpio/0"
        proc_http
    END IF
! Lese Status
    IF ctrl = Status THEN
        http_string = http_string & "gpio/read"
        proc_http
    END IF
! Lese Vorlauftemperatur
    IF ctrl = VL_Temp THEN
        http_string = http_string & "temp"
        proc_http
    END IF
    IF ctrl = quit THEN
        STOP
    END IF
END SUB
```

What can techBASIC do?

As you might have guessed from the name, techBASIC has very many similarities to the old-school BASIC programming language. However, when it comes to the range of available functions there is no comparison to the original BASIC. The techBASIC Reference Manual [3] offers a complete description of what is available: as well as the usual mathematical and string functions (such as LEFT$), techBASIC sports a number of functions to support event handling. For example, it is possible to have a program respond to the press of a button on the screen.
There is also a range of functions to support BLE and WiFi, allowing communication with the outside world. BLE and WiFi are also the only two communications protocols approved by Apple: this is not the case for USB or Bluetooth 2.x. Thus BLE and WiFi are the two wireless protocols of interest if we are looking at connecting to external sensors for applications such as IoT or home automation.
WiFi is supported using the `Comm` class. Using these functions it is very easy to create a simple HTTP client running on the iPhone or iPad. As one example, I have developed code (OnOffTemp.txt) that allows you to control a boiler remotely via an ESP8266 HTTP server. **Listing 1** shows the part of the program responsible for dealing with inputs from the user interface; the complete program can be downloaded from the Elektor website [4].
There are also functions provided in techBASIC to read values from the device's internal sensors, such as its accelerometer and its GPS receiver. There are graphics functions that allow you to plot complex curves and other graphical elements, but we will not look at those in detail here.
Much more interesting to us (the iCing on the iCake?) is the comprehensive set of functions for building graphical user interfaces (GUIs). With just a few commands it is possible to create an impressive array of widgets just like those used in any commercial application: buttons, sliders, date selectors and so on. There is a demonstration on the website of the makers of techBASIC that shows all the available widgets on the display of an iPhone or iPad.

Example circuit

The example we are about to describe shows how to read data from climate sensors using techBASIC. The sensors can measure temperature, humidity, atmospheric pressure and UV intensity. The readings are sent to an iPhone or iPad, which runs a techBASIC program, over BLE. For BLE communications we will be using a BL600 module, which will surely be familiar to many Elektor readers. We will make use of two breakout boards: an Adafruit Si1145 breakout board for UV intensity measurement and a BME280 breakout board for the other three quantities. The whole thing will be powered from a CR2032 coin cell: in my prototype one cell has already given about ten months of continuous operation.

The circuit is shown in **Figure 1**. If desired, a 'USB to UART converter' can be attached at JP1 to allow programming and debugging of the BL600 module. JP5 allows configuration of the operating mode of the BL600: the details have already been discussed comprehensively in Elektor. In our application the jumper must be fitted in the 'AUTORUN' position. One special feature is the button S1: this, in conjunction with the software, is the key to the low power consumption of the device. Normally the device is in what is called 'deep sleep mode', where the current draw is very low indeed. Only when button S1 is pressed will the device wake up, read the sensors one by one, and transmit the readings over Bluetooth. After about two minutes the BL600 returns to deep sleep mode. The software for the BL600 ($autorun$.klima.sb) is available for download from the Elektor website [4].

Scanning for BLE devices

Now we turn to the other end of the connection, the reception of sensor readings using techBASIC on the iPhone or iPad. Describing all the code in this article would take up far too much space; much of it, however, is self-explanatory and can easily be understood with a little study. Instead, we will just take a look here at the few lines of code that are needed to set up a connection with a BLE device: see **Listing 2**.

The function `scanBLE` is called when the 'Scan' button in the GUI is pressed. If the scan is successful then the call-

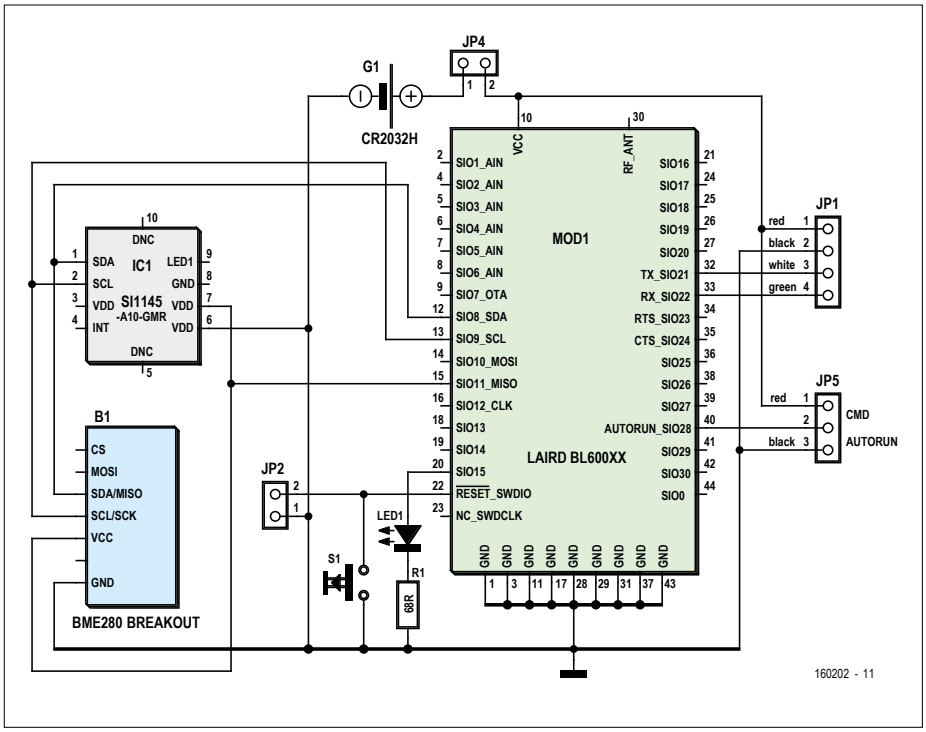

Figure 1. The external hardware includes the sensors and the BLE module.

Listing 2. Scanning for nearby BLE devices in the sensor application.

```
!-------------------- scan for BLE Device
------------------------------
SUB scanBLE
! Start the BLE service and begin scanning for devices.
    BLE.startBLE
    BLE.startScan(uuid)
END SUB

! Called when a peripheral is found. If it is a LAIRD BL600, we
! initiate a connection to it and stop scanning for peripherals.
!
! Parameters:
! time - The time when the peripheral was discovered.
! peripheral - The peripheral that was discovered.
! services - List of services offered by the device.
! advertisements - Advertisements (information provided by the
! device without the need to read a service/characteristic)
! rssi - Received Signal Strength Indicator
!
SUB BLEDiscoveredPeripheral (time AS DOUBLE, peripheral AS
BLEPeripheral,
                             services() AS STRING, advertisements(,)
AS STRING, rssi)
    PRINT peripheral.bleName
    IF peripheral.bleName = "LAIRD BL600" THEN
        bl600 = peripheral
        BLE.connect(bl600)
        BLE.stopScan
    END IF
END SUB
```

back function `BLEDiscoveredPeripheral` will be invoked. This function name, and indeed the names of all the other functions that are called during the process of setting up a connection, are fixed by techBASIC. There is therefore no need for you to specify the name of the callback function yourself, as is necessary in many other programming languages. In the next step the function `BLEPeripheralInfo` is called. This retrieves a list of the available services on the BLE device. If you do not have this information in the documentation of your BLE device, you can use a techBASIC program called 'sniffer' to help: this will display a list of all available BLE device services with their UUIDs.

If the desired service (in this case the 'Virtual Serial Port' service) is selected in `BLEPeripheralInfo`, then the 'characteristic' of the service will be requested. The characteristic will be received in the function `BLEServiceInfo` and, at the same time, the corresponding data will be requested from the BLE device using a 'Notify Request'. The data are in turn received in the function `BLECharacteristicInfo` and written to the appropriate variables to update the GUI.

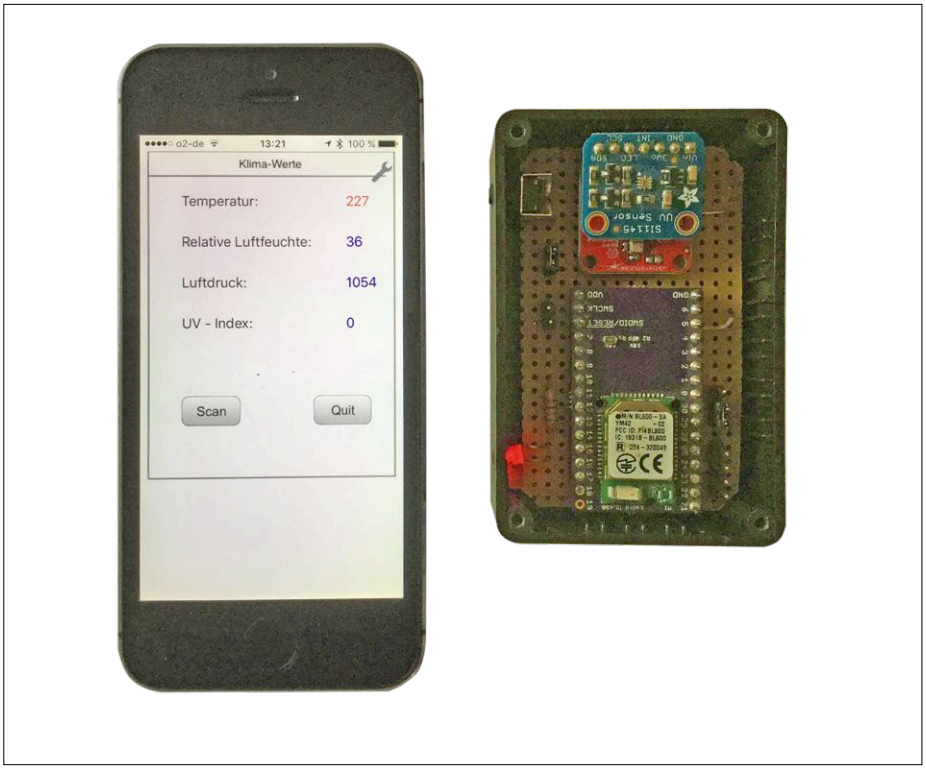

Figure 2. Data are received by the iPhone over BLE and shown on the display.

User interface

The function `showGUI` (see **Listing 3**) creates the various widgets that comprise the GUI, in this case just buttons and labels, and configures them to appear at the right positions on the display of the iPhone or iPad. The function is called once, when the program starts up.

It is worth noting that in no way did I write this program starting from scratch. It is simply a modification and extension of one of the many example programs that come with techBASIC. In this case I used the 'KeyFob' program as a template to start from. The program here will work on an iPhone as well as on an iPad: it includes the necessary adaptations to the coordinates of the graphical elements to suit each device. The program can determine what type of device it is running on, and so only one version of the code is required for the two devices. The complete program (klima.txt) is available for download from the Elektor website [4].

Listing 3. Definition of the user interface for the sensor application (excerpt).

```
!------------------------ showGUI -------------------------------

SUB showGUI
    x_lab = 50
    y_lab = 50
    y_distance = 50

    DIM tempLabel AS Label
    !tempvar = "20"
    tempLabel = Graphics.newLabel(x_lab, y_lab, 100)
    tempLabel.setText("Temperatur: ")
    tempvarLabel = Graphics.newLabel(x_lab+200, y_lab, 100)
    tempvarLabel.setText(tempvar)
    tempvarLabel.setColor(1,0,0)

    y_lab = y_lab + y_distance

    DIM humLabel AS Label
    !humvar = "30"
    humLabel = Graphics.newLabel(x_lab, y_lab, 200)
    humLabel.setText("Relative Luftfeuchte: ")
    humvarLabel = Graphics.newLabel(x_lab+200, y_lab, 100)
    humvarLabel.setColor(0,0,1)

    y_lab = y_lab + y_distance

    DIM airpLabel AS Label
```

The Android original

A few years ago it was obligatory to register with the Apple Developer Program in order to gain permission to transfer your own code onto your iPhone or iPad. The program cost around US$100 per year. For that reason I decided at that point to develop the app under Android (KLIMA.zip). As luck would have it, at the same time there appeared in Elektor a series of articles on the 'BL600 e-BoB' (starting in the March/April 2015 issue). One of the example applications was in temperature measurement. On the basis of the program presented there I was able to put together a suitable Android application, and in the process I did indeed get to know about Android and the 'Android Studio' development environment. However, the effort involved was considerable in comparison to the results achieved (reading values from a couple of sensors and sending them out over BLE). I must say, however, that the graphical interface did look a bit prettier than the techBASIC version (see **Figure 3**).

Conclusion

It is a pity that I did not discover techBASIC earlier, as it would have spared me my lengthy foray into the world of Android app development. In future I will also not have to carry two smartphones about, as all my Bluetooth applications will now run on the iPhone. However, it does require me to remove all the Bluetooth chips from my existing projects and replace them with Bluetooth 4.0 (BLE) equivalents; but that should not be an enormous effort. Any readers interested in studying this topic in more depth are recommended to read the book 'Building iPhone and iPad Electronic Projects: Real-World Arduino, Sensor and Bluetooth Low Energy Apps in techBASIC' [5].

(160202)

Web Links

[1] www.byteworks.us/Byte_Works/techBASIC.html

[2] https://itunes.apple.com/us/app/techbasic/id470781862?ls=1&mt=8

[3] www.byteworks.us/Byte_Works/Documentation_files/techBASIC%20Manual%203.3.1.pdf

[4] www.elektormagazine.com/160202

[5] http://shop.oreilly.com/product/0636920029281.do?sortby=publicationDate


```
    !airpvar = "40"
    airpLabel = Graphics.newLabel(x_lab, y_lab, 200)
    airpLabel.setText("Luftdruck: ")
    airparLabel = Graphics.newLabel(x_lab+200, y_lab, 100)
    airparLabel.setColor(0,0,1)

    y_lab = y_lab + y_distance

    DIM uviLabel AS Label
    !uvivar = "50"
    uviLabel = Graphics.newLabel(x_lab, y_lab, 200)
    uviLabel.setText("UV - Index: ")
    uviparLabel = Graphics.newLabel(x_lab+200, y_lab, 100)
    uviparLabel.setColor(0,0,1)

    !Create a scan button.
    scan = Graphics.newButton(but1_pos ,300)
    scan.setTitle("Scan")
    scan.setBackgroundColor(1, 1, 1)
    scan.setGradientColor(0.7, 0.7, 0.7)

    !Create a quit button.
    quit = Graphics.newButton(but2_pos ,300)
    quit.setTitle("Quit")
    quit.setBackgroundColor(1, 1, 1)
    quit.setGradientColor(0.7, 0.7, 0.7)

END SUB
```

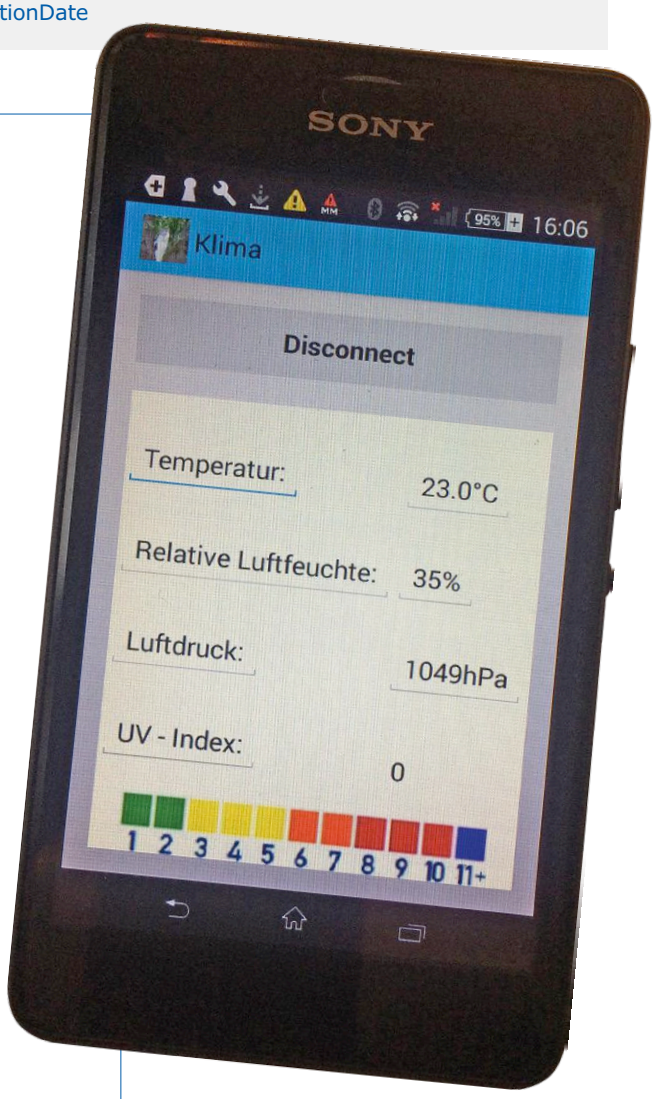

Figure 3. For comparison: the example application running on Android.