IN DEPTH Android apps

Write your own apps in Basic for Android

Darien Graham-Smith introduces a powerful visual programming environment that makes it easy to create apps for Android

Android apps are big business, but the native development tools aren’t exactly beginner-friendly. There’s a laundry list of technical concepts you need to master before you can even write a simple “Hello world!” program. You also have to learn to work with Eclipse, an open source development environment with plenty of quirks.

Happily, you don’t have to use native tools. Absolute beginners can assemble their own programs in a visual environment using the free App Inventor service, which we introduced in issue 204 (see http://appinventor.mit.edu). For those seeking a more hands-on approach, it’s possible to create, share and publish apps using a friendly Basic-type language, without having to deal with the more abstruse aspects of native Java development.

Basic options

One free option is RFO Basic for Android, an open source programming environment that’s free to download from Google Play. It’s modelled on “classic” Dartmouth Basic, so it’s very easy to get up and running. However, since the package runs entirely on your smartphone or tablet, you don’t get the conveniences of a Windows-based development environment – nor, with most devices, a physical keyboard.

We’ve therefore chosen to focus instead on Basic for Android (B4A), a Windows-based package that’s modelled on Visual Basic (VB). It isn’t quite as simple as RFO Basic, but it’s much easier than diving straight into Eclipse and Java – and much closer to the native Android way of doing things, giving you a leg-up if you choose to graduate to native programming later on.

Setting up Android

Basic for Android runs within Windows, but Android programs won’t work on your desktop – they must be tested on an Android device. The trial version of B4A includes a feature called “remote compilation”, which beams your code to a mobile phone or tablet for testing. At present, though, this works only for simple programs and isn’t in the full version of B4A.

A more flexible way to test your code is to compile it locally and run it in an Android Virtual Device (AVD) – an emulator, in other words. This is slower than using a real device, but it lets you test your code at any resolution. To set this up, you’ll need to install the Java Development Kit and the Android SDK; our walkthrough on p30 shows you how.
Compiling and running your first Android program

Once you've installed Java and the Android SDK, and created your AVD for testing, it's time to configure Basic for Android.

If you're using the trial version of B4A, the IDE opens containing a placeholder program, which simply pops up a message box. We can run this program to test everything is working — but first we need to direct B4A to the resources it needs for compiling Android apps.

To do this, check that Remote Compilation Mode is unticked under the Tools menu, then select Tools > Configure Paths. You'll be prompted to provide paths to the needed Java and Android components.

If you're using 64-bit Windows, look in the Program Files (x86) folder rather than the main Program Files folder as suggested. Leave “Additional libraries” blank for now.

Now you're ready to try compiling and running the sample program. Hit the blue “Run” icon on the toolbar (or select Compile & Run from the Project menu).

The first time you do this you'll be prompted to provide a filename, and then a Package Name — a unique identifier for your app, which is structured as a sort of backwards URL, such as uk.co.pcprou.firstapp. It isn’t necessary to actually register this as a real subdomain; the idea is simply to come up with an identifier that's guaranteed not to clash with an existing one. For more on package naming, see www.pcprou.co.uk/docs/2200 apa.

Finally, you'll be asked for an application label — the name that will appear below the app's icon in the Android app menu. The program should now compile, and B4A will try to install it to a connected Android device, so make sure your AVD is up and running. You can also test on an actual smartphone or tablet if you enable Debugging mode — see the B4A documentation for more information about this.

If all is well, you should see the message box appear on the screen of your AVD. Click on it and it will close. Congratulations! You've compiled and run your first Android app. Next time you run it, all of the settings you just entered will be remembered, so it will compile and run straight away. If you want to change the package name or application label, you'll find the options under the Project menu.

Note that Android apps don't shut down automatically (unless you write code to make them do so), so if you're running in Debug mode you'll have to stop execution by clicking the green Stop button in the debug panel at the bottom of the IDE (see Debug mode, below).

If you're using the full version of B4A, the process is much the same, except that the first time you open the IDE you'll be prompted to register. One user is permitted to run B4A across two computers, but be warned that the licence file is deleted after registration, so keep a copy of it somewhere safe.

You'll also notice that the full version of B4A opens with an empty program framework. If you want to test that everything is working as above, add a single line of code so that lines 14-16 of the framework read as follows:

```sub
Activity_Create(FirstTime As Boolean)
MsgBox("This is my first Android app!", "Information")
End Sub
```

As you type, your code is automatically colour-coded and capitalised to help you follow what's going on. Once you've entered this single line, you can compile and run the program, and click on the message box (you'll see the second string, “Information”, is its title) to see that everything is working as it should.

The structure of a B4A program

As we've mentioned, B4A is very similar to VB. Programs don't proceed linearly from the first line to the last; instead, they're event driven. You build up an app by creating standalone sections of code to handle specific events, such as when a particular button is pressed.

Debug mode

At the top of the B4A development window you'll see a dropdown menu offering three different compilation modes. While working on your program, it's a good idea to keep this set to “Debug”. In this mode, the development environment “watches over” what's happening on the device while your code is running. If something isn't working, you can follow the program flow and keep an eye on the values of variables as it executes, to see where things go wrong.

You do this by setting one or more breakpoints in your code — points where execution will be automatically suspended for you to take a closer look. To set or remove a breakpoint, click to the left of the relevant line number in the main development environment. Once your program hits that point, the debug pane will show the values of all variables. You can click the green “play” button to resume execution, or press the "step" button to advance through the program one line at a time. Note that you can't edit your code while the debug view is active.

The other compilation modes are “Release” and “Release (obfuscated)”. These create a smaller, more efficient package that can be uploaded to Google Play. Choosing obfuscated mode randomises the names of variables and sub in the object code, making it more difficult for a hacker to reverse-engineer your work.
Setting up Java and the Android SDK

1. The first step is to download and install the Java Development Kit (JDK) from www.pcpro.co.uk/links/220ida2. Choose the Windows x86 version, even if you’re using 64-bit Windows, since this is what the Android SDK and B4A expect. Accept the default installation options – there aren’t many – and let the installer do its thing.

2. Next download and install the Android SDK: this comes from www.pcpro.co.uk/links/220ida3. This contains the basic components needed for Android development. Once it’s installed, run the SDK Manager, which allows you to download the resources required to develop and test apps for the various versions of Android.

3. We don’t need to target the latest version of Android – version 2.2 (Froyo) supports all the features our app requires. Deselect any pre-ticked components, click to expand the 2.2 tree, tick the SDK Platform and click “Install packages”. If you want to add other components later, you can re-run the SDK Manager at any time.

4. Finally, let’s set up our Android emulator. Launch AVD Manager, click New and type in a name for your virtual device. Click Target and choose Android 2.2 as your operating system. Default settings should appear for this OS version: you can safely accept these. Click Create AVD, then Start… to boot up your virtual Android device.

pressed. These sections of code are called “subs”, just as they are in VB.

The framework that opens when you first open B4A is pre-populated with several subs. The one containing the code that makes the message box appear is called “Activity_Create” (don’t worry about the part in brackets for the moment). “Activity” is Android jargon, referring to the application’s full-screen graphical interface. The suffix “_Create” specifies that this sub is triggered when the activity first appears. In other words, code placed here will execute automatically when the program runs.

You can write your own subs to handle other events. As an example, let’s write some code to place a button onscreen, and a sub that will be called when it’s clicked. We’ll start by editing Activity_Create so it reads as follows:

```vbnet
Sub Activity_Create(FirstTime As Boolean)
    Dim myButton As Button
    myButton.Initialize("myButton")
    myButton.Text = "Click here to open the message box"
    Activity.Addview(myButton, 29, 100, 440, 60)
End Sub
```

B4A uses familiar object-oriented syntax, with objects having methods and properties that can be accessed by appending their names to the object name. As you’ll have noticed as you typed in the above code, when you type a dot, a dropdown menu appears showing valid keywords; when you type an open bracket, you get a helpful pop-up reminder of the required syntax.

In the aforementioned code we declare a new notional button called myButton, then initialise it – a necessary step just to tell Android we’re actually going to use it. The parameter in brackets specifies the name of the event we want to generate when we click on it.

Next we give the button some text, and finally, we need to put the button onscreen. In Android, all visible interface elements are called “views”, so the method you use to add a button to an activity is “Activity.Addview”. The parameters at the end specify which button we’re talking about, and give its desired position and dimensions.

Now we can write an event handler for our button. This is as easy as adding a new sub to the program like this (make sure you don’t add it within the bounds of an existing sub):
**Homework: a working calculator**

Once you’ve typed in our very simple Android app, you’ll probably want to try something more ambitious. A possible next exercise might build on the principles we’ve covered to develop a working calculator. This will involve some new techniques, but nothing difficult, and if you get stuck you can always refer to the online documentation at www.pcpco.co.uk/links/220dat.

- In the Designer, create a set of buttons, showing digits from zero to nine, plus mathematical operators for add, subtract, multiply and divide, a “clear” button and an “equals” button.
- Create a text label that will be the “screen” of our calculator.
- Import your layout into a new program and create event handlers for the number buttons, to append digits to the end of the label’s Text property (use the & operator to concatenate strings).
- Create event handlers for the operator buttons; you’ll want to create new variables to hold the first operand while the user enters the second, and to remember which operator was pressed.
- Create an event handler for the “equals” button that performs the relevant operation, and outputs the result to the label.
- Create a handler for the “clear” button so the user can start a new calculation.

There are plenty of ways to build on this further. For example, you could use B4A’s `length` and `Substring2` text properties to add a “backspace” button. Or you could throw the whole thing away and write something that’s of more interest to you. The sky’s the limit, and the best way to learn is by following your own curiosity.

For now we’ll keep things simple. Create an interface with one button and one label, and arrange them however you like. Then go to the Tools menu and select “Generate Members”. Assuming you’ve kept the default view names of Button1 and Label1, you’ll see a window open showing expandable, selectable headings for Activity, Button1 and Label1. Tick Button1, then expand it and tick Click. Do the same for Label1. Click “Generate Members” and a framework for initialising these views and handling the selected events will automatically be created and inserted into the code view.

Now click the close button in the Designer and you’ll be asked to save your layout. Give it a name with no spaces and return to the code view. You’ll see that the Generate Members operation has also inserted two new Dim statements in the “Globals” sub, setting up these views to be accessible from anywhere in the program. You’ll also see two new subs at the bottom of your code for handling click events.

The new layout won’t be automatically incorporated into our program. To replace our old, manually defined layout with the new one from the

**“A complex layout can be put together using B4A’s built-in graphical designer”**

other details: you can make precise adjustments by typing in the desired text, and you can also edit properties such as text labels. If you click to expand the Text Style options, you can control text size, colour and alignment. If you want to change the colour of a button, look for a property called “Drawable” and change this from “DefaultDrawable” to “StateListDrawable”. This will expose more options for controlling the button’s appearance.

It’s worth spending some time playing with the Designer, as it’s a powerful utility and a good way to learn the names and appearances of the standard Android interface elements. Under the Tools menu you’ll find options to duplicate selected views and adjust grid settings. You can also use Designer Scripts to create dynamic designs that automatically adjust to the size and orientation of the user’s device – but we’ll deal with that next month.