
LiveCode is a tool for developing mobile apps designed for users who don’t want to use Objective-C, C++, or Java. LiveCode Mobile Development Beginner’s Guide, Second Edition will explain how to create apps and upload them to the app stores with minimal effort.

You will begin with a simple calculator application and quickly enhance it using LiveCode Mobile. You will also learn about the interface controls for videos and images of LiveCode’s environment. You’ll go on digging into configuring devices and making rich media applications, and then finish by uploading your mobile applications to app stores. You will learn how to build apps for iPhone and Android devices with LiveCode Mobile through sample applications of increasing complexity.

What this book will do for you...

- Create a simple sample application and build its interface
- Write code using a multimedia scrapbook as an example application
- Make a “To do/reminders” application
- Upload your final app to the app stores
- Create a jigsaw puzzle app that takes advantage of several mobile device features
- Make standard-looking buttons and fields and programmatically create the screen layout
- Preview LiveCode version 8 Widget and Builder capabilities

The Beginner’s Guide approach...

- Clear step-by-step instructions for the most useful tasks
- Learn by doing, start working right away
- Leave out the boring bits
- Inspiring, realistic examples give you ideas for your own work
- Tasks and challenges encourage experimentation


LiveCode Mobile Development

Beginner’s Guide

Colin Holgate  Joel Gerdeen

$ 49.99 US
£ 32.99 UK

“Community Experience Distilled”

Learn by doing: less theory, more results

Create interactive mobile apps for Android and iOS with LiveCode

Beginner’s Guide

Free Sample
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 3 'Building User Interfaces'
- A synopsis of the book’s content

About the Authors

Colin Holgate was originally trained as a telecommunications technician in the Royal Air Force, but with the advent of the personal computer era, he transitioned to working as a technical support engineer for companies, which included Apple Computers, UK. In 1992, he moved to the US to become a full-time multimedia programmer working for The Voyager Company. In that role, he programmed several award-winning CD-ROMs, including A Hard Day's Night and This Is Spinal Tap.

For the last 17 years, Colin worked for Funny Garbage, a New York City-based web design company. In addition to using Adobe Director and Adobe Flash for online and kiosk applications, he has used LiveCode to create in-house and client production tools. At the RunRevLive Conference in 2011, Colin entered and won a contest to create a mobile application made with LiveCode.

Joel Gerdeen obtained a PhD in engineering mechanics and biomedical engineering from Iowa State University, where he started using computers in experimental research. In his first employment as a structural analyst, he developed software to assist other engineers to graphically model heavy machinery. His support of engineering computer usage transitioned into a career of software project management at FMC, Honeywell, and BAE Systems, all of which were Fortune 100 companies. Joel has experienced computing evolution from loading machine code through switches on a DEC minicomputer to booting a Raspberry Pi from a microSD card. He has worked with microprocessors, timesharing, personal computers, mainframe business systems, and latest mobile devices.
After 35 years of employment, Joel ventured into mobile software development in 2010, working with a small start-up company and publishing numerous apps on both Apple and Google app stores. After working with separate iOS and Android development environments, he discovered LiveCode and was able to build on his former HyperCard experience. Joel is also active in the mobile development community in Minneapolis and has presented LiveCode at local conferences.
Everyone you know has a smart mobile device of some kind. You probably own several too! The general idea of having utility applications on a phone is not new. Even cell phone and PDA games have existed for years, but the way that iPhone used touch instead of a stylus or keyboard and the way it used gestures to reduce the number of steps to do something was a game changer.

iPhone was released in June 2007 and the Android OS was released in September 2008. If you wanted to create something that worked on both platforms, you'd had to learn two development environments and languages: Objective-C for iPhone and Java for Android.

In the desktop world, there are several development tools that allow you to publish apps on both Mac and Windows as well as Linux in the case of LiveCode. The most successful of these tools are Adobe Director, Adobe Flash, Unity, and LiveCode. Publishing apps to iOS was introduced with Adobe Director 12, which means that all four tools are also suitable for mobile development.

These tools have different strengths; in some cases, the strengths relate to the nature of the applications you can make and in other cases, they relate to how accessible the tool is to people who are not hardcore programmers. If you want to make a high-quality 3D game, Unity would be the best choice, with Director and then Flash as other choices. If you need a lot of character animations, Flash would be the best choice, Adobe Director being a good alternate.

If the most important thing for you is how approachable the tool is, then LiveCode wins easily. It's also a valid choice to make the majority of apps you might wish to make. In fact, for apps that are a set of single screens, as would be the case for most utility apps as well as board and puzzle games, LiveCode is better suited than other tools. It also has better access to native interface elements; with the other tools, you usually have to create graphics that resemble the look of native iOS and Android controls instead of accessing the real thing.

With its easy-to-use near-English programming language and the stack of cards metaphor, LiveCode lets you concentrate more on creating the app you want to make and less on the technicalities of the development environment.
What This Book Covers

Chapter 1, *LiveCode Fundamentals*, introduces you to the LiveCode environment and to its near-English programming language. Experienced LiveCode users can skip this chapter, but for someone new to LiveCode, this chapter will take you through the process of creating a simple calculator app as a way to make you familiar with the various tools and hierarchy of LiveCode.

Chapter 2, *Getting Started with LiveCode Mobile*, describes in detail how to set up your Mac or Windows computer so that you are ready to develop and publish mobile apps. This chapter will take you all the way through from signing up as an iOS and Android developer to creating and testing your first LiveCode mobile app.

Chapter 3, *Building User Interfaces*, shows how to use some of the standard mobile features, such as date pickers, photo albums, and a camera. This chapter will also show you how to make your own buttons that have an iOS-like look to them and how to use the LiveCode add-on, MobGUI, to make your life easier!

Chapter 4, *Using Remote Data and Media*, discusses the structure of your apps, where to place your code, and how to read and write to external text files. Here, we will also create a mobile app that is a "web scraper" capable of extracting links and media from a web page to show or play media from that page.

Chapter 5, *Making a Jigsaw Puzzle Application*, will show you how to process image data and how to use the information to create a color picker, detect regions, and to make a collision map. We will then create a full jigsaw puzzle application that takes its image from the photo album or device camera.

Chapter 6, *Making a Reminder Application*, examines which information is needed to represent a "reminder" and how to set up notification events so that you are alerted at a specified date and time. Here, we will make a reminder app that can create a list of such events and even list those events based on your current location.

Chapter 7, *Deploying to Your Device*, is a reference chapter that describes all of the mobile publishing settings. This chapter also shows you how to send apps to beta testers and how to get started with the submission of your finished app to various app stores.

Appendix, *Extending LiveCode*, describes add-ons to LiveCode that will make your mobile apps look better or will extend the mobile capabilities of LiveCode. The planned LiveCode builder and widget capabilities of LiveCode's Version 8 are introduced as well.
Chapter 3

Building User Interfaces

So many different screens!

When making utility or game applications for desktop computers, you can often get away with having a particular sized window for which you can make custom graphics that exactly fit. With mobile devices, you have to cope with a wide range of screen sizes and aspect ratios and also have to interface elements that look correct for the operating system on the user’s device.

LiveCode is capable of publishing on Mac, Windows, and Linux and goes some way toward solving the difficulty of making interface elements look right for each platform. The View menu has a Look and Feel menu item where you can choose between Native Theme, Mac OS Classic, Windows 95, and Motif. The same isn’t true for mobile operating systems as all controls look like Motif. You still have two choices though: you can create graphics that look like they belong in your target OS, or you can call native routines in order to let the system itself present the appropriate controls.

In this chapter, we will:

- Set up a test bed mobile application
- Open an e-mail and browser windows
- Show a date picker control
- Load pictures from the library and camera
- Make an iOS styled button
- Manually lay out an interface
- Use code to lay out an interface
- Look at a powerful mobile interface control’s add-on
Setting up a test bed mobile app

As a proving ground for the things we’re going to try, we'll set up a single mobile app that has multiple screens, one for each of the things we want to test.

What should we call the test bed app? We could call it almost anything, but we'll let the iPhone make the decision for us. On the iPhone and iPod touch, there is only a small amount of space under the home screen icons for the name to appear. iOS will take your nice long app name and show a shortened version of the name, using ellipses to concatenate the ends of the name together. My super duper app will appear as My sup...app, not quite as informative! The number of letters that can appear without the text being truncated will vary depending on the width of the letters used, but typically, it has a limit of 11 letters. So, we will call the test bed app LC Test Bed, which is exactly 11 letters!

Time for action – making the test bed stack

Before we create the iOS and Android apps, we should get what we want ready, as a LiveCode stack and fully test it on our desktop computers. The following steps are going to assume that you know how to do what is asked in LiveCode, without precise instructions.

1. Open LiveCode, create a new Mainstack, and save it as LCTestBed.
2. Set the screen size to 320 x 480. This is just to make sure that things appear on the smallest of screens. The things we will make, will appear at the upper-left corner area of the larger screens.
3. We are going to make a button for each card in the stack; let’s start by making a card named Menu.
4. Add buttons for Email, Browser, DatePicker, and Picture. Make sure that the buttons are big enough to touch on your devices. You should have something like what is shown here:
5. Create four new cards and name each one so that they match the button names.

6. Back at the first card, set the script of each button to go to the matching cards with this script:

   on mouseUp
   go card the short name of me
   end mouseUp

7. On each card, create a button to return to the Menu card. Name the button Menu. Set its script to the same as the other buttons.

8. Select the Run (browse) tool and try clicking on the buttons to jump to the four cards and back to the menu.

What just happened?

Well, nothing too exciting! However, you should now have five cards and the ability to go in and out of the Menu card. We're going to add scripts to each card to help illustrate its various features. The most efficient approach will be to add all the scripts and related buttons and fields and then to test the final test bed app in one go. However, where's the fun in that! Instead, we'll go one feature at a time...

Invoking the desktop e-mail application

There are many cases where you may want to hear from the users of your applications. Perhaps, you want them to e-mail suggested improvements or to ask you questions. You could easily launch their e-mail program and leave the user to figure out what to write. Alternately, you could set the To address, Subject, and even some of the Body of the message. At the very least, it would make your life easier because you could filter incoming e-mails based on something that you placed in the Subject field.
Time for action – calling the native e-mail application

In the following steps we’ll make some fields and a button to try sending an e-mail feature:

1. Go to the Email card and create four fields. Name them To, CC, Subject, and Body.
2. Make a button named Test.
3. In the Test button, add this script:
   ```
   on mouseUp
       put field "To" into toText
       put field "CC" into ccText
       put field "Subject" into subjectText
       put field "Body" into bodyText
       revMail toText,ccText,subjectText,bodyText
   end mouseUp
   ```
4. Select the Run tool and type in example information in each of the fields.
5. After setting up the Standalone Application Settings... and selecting the Test Target, click on the Test button.

What just happened?

One neat thing about the LiveCode syntax is that the code for mobile also works for desktop applications and vice versa. All being well, when you click on the Test button, you will find yourself in your default e-mail application ready to send the message that you had entered in the LiveCode stack fields.

Installing the e-mail test on devices

It’s no great surprise that the desktop test worked. The ability to open other applications is the basic feature of LiveCode. Still, it’s neat to send over some initial text for the new message to take on. Next, we should check whether this works on devices too.
Time for action – trying the test bed stack on devices

Connect your Android and/or iOS device to your computer using USB. These instructions are almost the same as in the previous chapter, when we tested a “Hello World” stack. After this point, any directions will be briefer and based on the assumption that you know the steps needed to test an app on your device. Chapter 7, Deploying to Your Device, describes all the options in the Standalone Applications Settings dialog. For the moment, we’re only going to fill in a few details, so here, we will just view a portion of the dialog, starting with the Android settings:

1. Make sure that Android is checked in the Standalone Application Settings dialog.
2. In the Identifier field, type in an identifier that will be unique; com.yourname.lctestbed would do.
3. Get your device to its home screen past the initial lock screen if there is one.
4. In LiveCode, choose Development/Test Target and select your Android device. It will be named as Android followed by a long number.
5. Choose the Development/Test option.
6. After compilation, the stack should run on your Android device and you should be able to touch the Email button and perform a test message that will use the Android e-mail application.
On iOS, if you haven't already done so, read Chapter 7, Deploying to Your Device, on how to deploy to your device. At least read the parts that show you how to install your iOS developer certificates and provisioning files. As with Android, we're only going to alter a couple of items in the Standalone Application Settings. The following is the screenshot of the dialog that we'll be altering:

Perform the following steps for an iOS device:

1. Change the **Standalone Application Settings** to **iOS**.
2. Under **Basic Application Settings** of the iOS settings is a **Profile** drop-down menu of the provisioning files that you have installed. Choose the one that is configured for the device you are going to test the app on.
3. In the **Internal App ID** field, type in a unique ID. As with Android, `com.yourname.lctestbed` would do. `yourname` would of course be your name or your company name.
4. If you are testing on iPad, select the **iPod, iPhone and iPad** option from the **Supported Devices** drop-down menu.
5. Close the dialog and choose **Save as Standalone Application...** from the **File** menu.
6. When the saving is done, you may see a warning message telling you about missing splash screens and icons. It won't matter for now.
7. In **Finder**, locate the folder that was just created and open it to reveal the app file itself.
8. Open **Xcode** and choose **Devices** from the **Window** menu.
9. You should see your device listed. Select it and if you see a button labeled **Use for Development**, click on that button.
10. Drag the app file straight from the Finder window to your device in the Organizer window.

11. The small colored circle next to the device will turn orange for a moment and then back to green.

12. You can now open the app and try the Email button and test message, which will use the standard iOS Mail application.

What just happened?
We went through the steps needed to install the test bed app on both Android and iOS devices. We also had to change a couple of things in the standalone application settings. As you saw, there are quite a lot of settings in there. You can look forward to learning about them all in Chapter 7, Deploying to Your Device!

Opening a web page
Another requirement in your applications is being able to present additional online information. You want the user to click on a link, or touch as the case may be, so that he/she is taken to a page that lists all the other applications that the user can buy from you!

Time for action – calling the native browser application
This next test will go faster, or at least, the instructions will be briefer, as we will condense some of the steps in more concise directions, as given here:

1. Copy the Test button on the Email card and paste it on the Browser card, just to save you some time making the button look nice.

2. Edit the Test button script and change it to this:
   ```javascript
   on mouseUp
     launch url "http://www.runrev.com/
   end mouseUp
   ```

3. Choose the Run tool and click on the Test button. You will see the RunRev home page in your default browser.

The steps for trying the app on devices is exactly the same as with the steps to test the e-mail feature. For Android:

1. Select Android in the Standalone Application Settings.

2. Select your Android device as the test target from the Development menu (most likely, it will still be selected from before).
Building User Interfaces

3. Select Test from the Development menu.
4. The previous test of the app will be overwitten and the new version will be launched automatically.
5. Try clicking on the Browser button and the Test button that you just created on the Browser card. The http://runrev.com/ page should be opened if you click on them.

For iOS:

1. Select iOS in the standalone application settings.
2. Redo Save as Standalone Application and then drag the app file on your device in the Organizer window of Xcode, as you did the first time.
3. Try the Browser and Test buttons; you should see that the RunRev home page has opened inside Safari.

What just happened?
As with the Email test, adding the standard code to open a web page works for Android and iOS just as it does for a desktop computer.

If you are testing on both Android and iOS, you will notice that the behavior is different when you return after looking at a web page. With Android, you can press the back arrow button and still be on the Browser card of your stack. With iOS, the stack is restarted when you return. We will examine a solution later, where we write data to an external file, so that when the app is reopened, we can return the user before leaving the app.

The mobile-only date picker
The next couple of examples we will try are the ones that only work on mobile devices and not on desktop computers.

Time for action – displaying a date picker
Many applications require the user to choose a date for an event, and with mobile devices, there is a particular look to the date picker that you are shown. Using LiveCode let's us display such a control:

1. Copy the Test button from the Browser card and paste it on the DatePicker card.
2. Edit the script to make it like this:
   on mouseUp
   iphonePickDate "date"
   end mouseUp
3. Select the Run tool and try the Test button. You'll see an error because this is a mobile-only feature.

4. For a change, select iPhone or iPad Simulator from the Development/Test Target menu and then choose Test from the Development menu.

5. You will see your stack open in the iOS simulator and you can try the DatePicker and Test buttons to then see the iOS date picker displayed.

6. Perform the same old Save As and install using the Organizer window steps to try the date picker on your iOS device.

7. Touch the DatePicker button on the menu card and the Test button on the Datepicker card. An iOS native date picker should appear.

**What just happened?**

Hopefully, you're getting better and are able to build and install mobile apps faster by now! In addition to testing again on a device, we also tried out the simulator. Generally speaking, it is faster to use the iOS simulator whenever you can, and only test on a device when you're checking things such as multi-touch, accelerometer, and camera support.

**Time for action – loading pictures for a mobile device**

Maybe one day it will be possible for us to bring in images from the user's desktop computer photo application or from their web camera, but for now, these features only work on mobile devices.

LiveCode can call upon the native photo library and camera apps. We will test both of these on Android and iOS, but of course, only if your device has some saved images and a camera. For Kindle Fire, which doesn't have a camera, make sure that you save some pictures in the Gallery app, so that we can at least try loading those. Follow these steps to load pictures for a mobile device:

1. Copy the Test button from the DatePicker card and paste it twice on the Pictures card. Change the name of the buttons to Test Camera and Test Library.

2. Edit the script of the test camera button to be:
   ```plaintext
   on mouseUp
   mobilePickPhoto "camera"
   end mouseUp
   ```

3. Edit the script of the test library button to be:
   ```plaintext
   on mouseUp
   mobilePickPhoto "library"
   end mouseUp
   ```
4. As we test the loading of pictures, the image that is loaded will lie on top of the test buttons, stopping us from returning to the menu card. To solve this issue, add this to the card script:

```
on mouseUp
if word 1 of the target is "image" then delete the target
end mouseUp
```

5. Go in the Standalone Application Settings and select Android.

6. We have to ask the Android OS permission to use the camera and store the image, so check the boxes for Camera and Write External Storage:

```
<table>
<thead>
<tr>
<th>Application Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Write External Storage</td>
</tr>
<tr>
<td>☐ Internet</td>
</tr>
<tr>
<td>✔ Camera</td>
</tr>
<tr>
<td>☐ In-App Billing</td>
</tr>
</tbody>
</table>
```

7. Repeat the steps for testing on your Android device, or installing on your iOS device.

8. Once the app is running on your device, touch Pictures on the first screen and then Test Library. You should see typical OS-specific options to choose a picture from your library or gallery.

9. The picture you have selected is loaded on the card window and will fill most of the screen, obscuring our test and menu buttons. The card script we entered gives you the ability to touch the image in order to delete it so that you can then try another test.

10. Try the Test Camera button. You will see the OS-specific camera application and when you have taken a picture and touched the Use or Ok button in the camera application, the image will be placed on the Pictures card.

**What just happened?**

These simple scripts illustrate how LiveCode is able to call the OS-specific applications to do what would otherwise take a lot of coding. What’s more, as later iOS and Android OS versions are released, the same simple scripts activate the more advanced features that Apple and Google will have implemented.
Pop quiz – getting the big picture

Q1. We take so much for granted when it comes to improvements in technology. You might feel hard done by if your phone’s camera is a measly 2 megapixels, but think back to how things were long ago and how big a picture you were used to seeing. In terms of the number of pixels, how many original Macintosh screens can fit in the area shown by a single 8 megapixel photo?

1. 4
2. 15
3. 24
4. 45

Answer: 45!

The original Mac had a screen that was 512 x 342 pixels. This will fit more than 45 times in the area of an 8 megapixel photo.

Making OS-styled buttons

It’s nice that LiveCode can call upon an OS’s native controls, but this raises a problem because the standard Motif-styled buttons will look ugly when used with the OS buttons. We can fix this either using built-in features of LiveCode or with the use of an add-on product.

Using bitmaps

As we saw in Chapter 1, LiveCode Fundamentals, you can use different bitmaps for the button’s states. You could get such images by taking screenshots of the buttons on your mobile device, at least with the iOS and Android OS v4 and later versions, or you can save a lot of time by downloading files that others have made available. Some of these files are only licensed for use in prototypes; here, we’ll take a look at one of the files that is also licensed to be used in commercial products.
Time for action – using Photoshop to prepare button states

The file we are going to use has Photoshop filter effects that other programs cannot handle, so unfortunately, you will need Photoshop to follow all of these steps or at least have a friend who has Photoshop! Pixelmator and GraphicConverter on Mac OS X can also extract graphics from the file, possibly, by just copying an area of the screen.

1. Read the following article:
   http://spin.atomicobject.com/2011/03/07/photoshop-template-for-ios-buttons/

2. The article points to some other sources of information; for now though, download the following file:
   http://spin.atomicobject.com/assets/2011/3/7/iOS_Buttons.psd

3. Open the file in Photoshop (it may open automatically).

4. In the Layers palette, hide the layers named Background and Tool Bar – Retina.

5. Expand the layer named Bar Button – Retina, and hide the Button Label layer.

6. Use the Marquee tool to select an area around the upper-right-hand side button. It should all look like this:
7. Choose Copy Merged from the Edit menu.
8. Select New from the File menu and make sure that the Background Contents property is set to Transparent and accept the size you are given.
9. Paste the content, it will be an exact fit, and you will see the idle state for that button.
10. Choose Save for Web & Devices... from the File menu.
11. In the save dialog, select 24 bit PNG and make sure that the Transparency box is checked. Save the PNG with a suitable name, say bluebuttonup.png.
12. Return to the main document and turn on the Visible = Active layer.
13. Do another Copy Merged | New | Paste | Save for Web & Devices....
14. Save the PNG as bluebuttondown.png.
15. Go back to LiveCode.
16. Reopen the test bed stack.
17. Use File, Import As Control, and Image File... to bring the two PNGs in the stack.
18. You can place the two images anywhere. Uncheck Visible in Basic Properties for each image.
19. Add a new button to the first card and give it the name Location.
20. Set the button script to:
   on mouseUp
     iphoneStartTrackingLocation
     put iphoneCurrentLocation() into theLocation
     answer theLocation["latitude"]
   end mouseUp

21. Select the Location button and in Basic Properties of the Inspector palette, turn off Show name and Opaque.
22. In Icons & Border, turn off Three D, Border and Hilite border.
23. Click on the magic wand button next to the Icon entry in the Inspector palette.
24. From the Image library drop-down menu, select This Stack.
25. Click on the lighter one of the two blue images.
26. Click on the magic wand button next to the Hilite icon entry and then click on the darker of the two images.
27. Resize the button just big enough to show the blue image without it being cropped.
28. Place a Label field on top of the button.
29. In Basic Properties, check the Disabled box. This is to make sure that the field doesn't trap the click you are going to perform. We want the button to get that click.

30. In Contents, enter Location.

31. Under Text Formatting, set the field to use Helvetica Neue, 18 point, Bold, and center aligned.

32. Under Colors & Palettes, set the text color white.

33. Align the field and the button so that the two are centered on each other.

34. If you now test using the iOS Simulator and click on the Location button, you will just see a zero, but trying on a device should display your latitude when you touch the button (you will have to give permission to the app to know your location the first time you press the button.)

Note that the example is in the iOS 6 format. iOS 8 can be found at:
http://www.teehanlax.com/tools/iphone/

What just happened?

Although the button we made, may not be of the perfect size or even have the correct look for a standalone iOS button, we did go through all of the steps that you would need to make button state images. Placing a LiveCode field over the image buttons doesn't necessarily give it the best appearance. In reality, you would take more time in Photoshop to make a button of the right width for the label you're using and might also have to add the text to the image itself. It would look better and would not need a field to show the button's name in LiveCode.

LiveCode is able to use code to create the images we need, by setting the points of a graphic and its fillGradient. However, once you have the component parts needed to simulate a button or other kind of control, it would still take a lot more scripting to manage these elements.

There is an easy way out, although, it will cost you $50!
Pop quiz – the cost of things these days

Q1. With the increase in your expectations about the size of a digital photo, you also expect to get a lot more for your money these days. While you weigh up the advantages of spending $50, how much better value do you think a computer’s memory is now, compared to 25 years ago?

1. 10 times better
2. Half as good
3. 100 times better
4. 20,000 times better!

Answer: 4

Yes indeed. 25 years ago, Apple was selling a 4 MB add-on kit for Macintosh II for about $1,500. They now sell a 64 GB add-on for Mac Pro for $1,200.

MobGUI to the rescue!

RunRev is based in Edinburgh, Scotland, and they’re a talented bunch! However, they’re not the only talented Scottish folk, there’s John Craig as well. He has developed a powerful add-on to LiveCode that includes an increasingly long list of iOS- and Android-OS-like controls. If you were to buy his product, you would have to pay $50 for which you get the current version plus any updates that are released in the 12 months following your purchase date. While we take a look at it here, we can also use a trial version of the product.

Time for action – getting started with MobGUI

As with the other add-ons to LiveCode, MobGUI needs to be installed in the LiveCode plugins folder. On Windows, this will be at `My Documents/My LiveCode/Plugins`. On Mac, it will be at `~/Documents/My LiveCode/Plugins`. This default location can be changed under the LC Preferences menu and Files & Memory. The following steps will guide you through getting started with MobGUI:

2. The .zip file will expand to become a LiveCode stack named MobGUI_V1-28.livecode for the current version. Hopefully, there will be a newer version when you download this.
3. Drag the stack into the plugins folder and reopen LiveCode.
4. Make a new Mainstack.
5. From the Development menu, choose Plugins/revMobGUI. This window will appear when you do so:

![MobGUI window](image)

6. The MobGUI window is much like a combination of the LiveCode Tool palette and Inspector palette.
7. Try dragging different items on the card and look at the options for each item.
8. The left-hand side list is scrollable and shows additional controls.
9. Once the controls are placed on the card, they can be manipulated like the normal LiveCode controls.

10. The selection field at the top right-hand side allows different themes to be selected. Select the android theme and click on Apply. Note that the controls you dragged out change. At the time of writing this book, the android theme represents the older pre-Lollipop themes.

11. After you have manipulated a stack, you can export the current theme for later use. You need to save the newly created stack named the MobGUI Theme stack. When opened later, this stack will have an Import button.

What just happened?
One remarkable thing about LiveCode is that the many windows and palettes that you use in the program are all just stacks, and we've started to make use of a rather specialized stack that is going to save us a lot of time and will give us a nice interface like the OS-specific interface.

A test bed app, the MobGUI way
We’re going to make much the same test bed app, but this time, we'll try to give a more iOS-like look to the app.

Time for action – using MobGUI to make a test bed app
As you work in LiveCode starting new stacks, closing others, and opening previously saved stacks, these actions can still occupy memory. Sometimes, you can get in a confused state where you're making a new Untitled stack only to find out there’s still an Untitled stack on the go, which you're asked about if you want to purge. So, why not treat yourself to a quit and start a fresh launch of LiveCode! The following steps will help you achieve that:

1. Create a new Mainstack. Set the name to MGTestBed and save it somewhere you can easily find it. Perhaps, in the folder with the LCTestBed stack, which was feeling lonely!

2. Open the MobGUI window by selecting Development/Plugins/revMobGUI.

3. In the MobGUI window's page of controls, select the Slider resizes box and move the slider to select a size of 320x480. This is the size of the original iPhone. Note the other sizes available. The card can also be resized with the LiveCode Inspector.
4. Select the **MobGUI context menu in IDE** option in the MobGUI window. This will enable you to edit the MobGUI control behaviors later.

5. Using the **Card Inspector**, set the name of this first card to Email.

6. Drag a **TabBar** on the card window. Click on **Snap to bottom** of the card window. It will also resize the width of the card.

7. In the MobGUI window, drag a button in the card window on top of the TabBar. Duplicate the button 3 times by holding the **Alt/Option** key and dragging it. Align the four buttons and distribute them across the card using LiveCode’s **Align Tools** in the **Inspector** palette.

8. Select each button and set their names and labels to Email, Browser, DatePicker, and Picture. Then, resize the buttons so that they fit their name text appropriately.

9. Select the Email button and choose **Object Script** from the **Object** menu or right-click on the button and choose the **Edit Script** option. The script will already look like the following screenshot:

10. Add the following **mouseUp** handler to the script

    ```
    on mouseUp
    go card the short name of me
    end mouseUp
    ```
11. You can copy the mouseUp script of the first button and paste the script in the other three buttons. Note that there is a `preOpenControl` handler created by MobGUI following the mouseUp script in each of these buttons. Do not change that!

12. We’ll need these elements on all of the four cards we’re going to make, so choose the Select All option and then choose Group Selected from the Object menu.

13. Make sure that the group is selected and in the regular LiveCode Inspector palette, check the Behave like a background box.

14. Make three more cards and name them Browser, DatePicker, and Picture.

15. From the LiveCode palette, drag a Label control on the card window for each of the four cards and set the name to match the card’s name.

16. In Standalone Application Settings, choose either iOS or Android, depending on the device you want to test on.

17. Set the Internal App ID or Identifier to `com.yourname.MGTestBed`.

18. If you’re performing the same in iOS, make sure that you choose a profile from the Profile drop-down menu.

19. You can now do a test from the Development menu, but first you have to choose either iPhone Simulator or your connected Android device.

What just happened?
It seemed like quite a few steps, but it doesn't take much time. We already have the navigation between the four cards and an authentic iOS-like interface.

Let's get some of the test features going, but in a more native, integrated way than before.

MobGUI native controls
One powerful feature of MobGUI is that it can use ordinary LiveCode controls as placeholders for what will become native controls when you run the app on a device. This isn't something that you can't do for yourself with code, but being able to move placeholder controls around, until you like the layout, would save a lot of time.

Time for action – using native controls from MobGUI
MobGUI allows you to switch between the native control theme for iOS and Android or to redefine your own themes.

1. Right-click on the little image of the iPhone in the MobGUI window and make sure that you're on the Native iOS controls set.
2. Go to the Email card and drag 3 Input text controls from the MobGUI window and one Multiline text control.

3. Name the Input controls as To, CC, and Subject and the Multiline text control as Body. You can also add some regular LiveCode Label fields alongside the input fields as an indicator of what to enter. Size the Body field big enough to enter a few lines of text. Also, add some background color to the fields or the card so that the fields are seen properly.

4. As you make each field, note that you can set the keyboard type as well. Set it to Email for the To and CC fields.

5. From the iOS Controls 1 set, drag two buttons on the card window. Name one Done and the other Send. You should have a screenshot like this after this step:
6. When we test the app and touch one of the fields, the keyboard overlay appears. We'll use the Done button as a way to hide the keyboard. Add a focus line to the mouseUp handler of the Done button script:
   ```
on mouseUp
    focus on nothing
end mouseUp
   ```

7. MobGUI can retrieve properties from these native fields using the mgText property. Change the Send button's mouseUp handler to use this property for each field and also, to call the revMail function:
   ```
on mouseUp
    put the mgText of group "To" into totext
    put the mgText of group "CC" into cctext
    put the mgText of group "Subject" into subjecttext
    put the mgText of group "Body" into bodytext
    revMail totext, cctext, subjecttext, bodytext
end mouseUp
   ```

8. Go to the Browser card.

9. From the MobGUI window, drag a Input control to the card window and name it URL.

10. Drag a Browser control to the card window and name it Page.

11. Adjust the sizes so that the text field fills the width of the card and the browser control fills the area between the text field and the tab bar at the bottom.

12. Select the Browser control and in the MobGUI window, enter a value for URL or use the default one already there. This will make the browser control load this URL as its first page.

13. Edit the script of the URL text field and add this handler, which looks for a Return key to go to the URL:
   ```
on inputReturnKey
    mobileControlSet "Page", "url", the mgText of me
end inputReturnKey
   ```

14. Try another test and go to the Email and Browser cards to see them in action.
**What just happened?**

We recreated the first two tests from our earlier test bed app, only now, it looks a lot nicer! Also, we made use of MobGUI's ability to get and set data in native iOS controls, in this case, using the mgText property and mobileControlSet.

Note that all MobGUI controls show up as **groups** in the LiveCode Inspector and as **Custom controls** in the Project Browser. These groups are made of customized LiveCode controls such as buttons, fields, and so on. MobGUI also adds a MobGUI card to the end of your stack. This card includes invisible buttons that have behaviors defined. Behaviors are methods to create common functionality between objects without duplicating the scripts. You can view these behavior scripts by clicking on the script button on the right-hand side of the Project Browser, when you display the MobGUI card. Unless you have a specific need to change these, just leave them alone.

**Have a go hero – other tests and pretty icons**

Go ahead and add the other two tests to the stack in the same manner as we did in the **Time for action** sections earlier in this chapter. For the DatePicker example, you could examine the **Dictionary** definition for **iPhonePickDate** to see examples of how to use the picked date data in the same manner as the previous sections for example, adjustments for different screen sizes.

So far, we have tested the size using the Portrait orientation with just an iPhone. You may want to use the same stack for iPhone and iPad or perhaps, iPad and an Android tablet, which have quite different aspect ratios.

Even if you just stick with the iPhone, you would still want to take care of Portrait and Landscape. We therefore, have to find ways to arrange the many controls on the card, so that they look their best on each screen size and orientation.

There are several ways to achieve this. First, we'll look at how to use a resize handler.

**Laying out using a resize handler**

When a stack's window size changes, LiveCode sends a `resizeStack` message that we can trap in order to rearrange controls for the new width and height.
**Time for action – a simple code layout example**

It could get quite complicated if you did lay out all of the card’s controls with code, so we’re only going to construct a simple case to show the technique. You can enhance this later for more complex cases.

1. Create a new Mainstack.
2. Add four buttons across the width of the card.
3. Put this handler in the card script:
   ```on resizeStack newWidth,newHeight
   put the width of button 1 into buttonWidth
   put (newWidth - 4 * buttonWidth)/5 into gap
   put the top of button 1 into buttonTop
   repeat with a = 1 to 4
       set the top of button a to buttonTop
       set the left of button a to gap + (a-1) * (gap+buttonWidth)
   end repeat
   pass resizeStack
   end resizeStack```
4. Resize the card window. The buttons should spread out evenly across the card.
5. Go to Standalone Application Settings and select the iOS option.
6. Make sure that the supported devices include iPad.
7. Set the orientation options to include all the four orientations.
8. From the Development menu, set the Test Target as the iPad Simulator and perform a Test.
9. In the simulator, choose either Rotate Left or Rotate Right from the Hardware menu.
10. The buttons should spread themselves out across the screen in both the portrait and landscape orientation.

**What just happened?**

In addition to making a simple example of how the resizeStack handler can be handled, we also saw that the changes in orientation also send the resizeStack message.
Laying out using the LiveCode Geometry Manager

While a control is selected on the card, the Inspector palette has an entry named Geometry. It's a somewhat strange interface! Let's take a look:

These faint horizontal and vertical bars are used to select whether you want the control to be scaled or positioned by a fixed amount or relative amount. That is, if a button is 100 pixels from the right of the card window and if you select the position a fixed amount away, then as you resize the card window, the button will remain 100 pixels away from the right edge of the window. If, on the other hand, you use the relative setting and the button is 80 percent across the card window, it will still be 80 percent across the window after you have resized it.

The first click on one of these bars will make it turn solid red in color and this indicates that it's a fixed amount away from the edge of the card. If you click on it again, it takes the shape of a red waveform, indicating that it's going to be relative.
In the screenshot, you can see that the selected button is set to a fixed amount from the bottom of the card and a relative amount from the right of the card. The image also shows the scaling settings for the control.

Note that an object can also be positioned relative to other objects. Refer to the Right object and Bottom object pop-up selectors in the preceding screenshot.

**Time for action – using the Geometry Manager to position buttons**

We’ll add some buttons to the stack that we are currently working on:

1. Take the first four buttons and duplicate them to get another set of four buttons below the previous ones.
2. Select the first of the new buttons and in the Geometry section of the Inspector palette, click once on the vertical bar and twice on the horizontal bar; you will end up with the state shown in the previous screenshot.
3. Do the same for the other three buttons.
4. Try resizing the card window.

**What just happened?**

That was quite a quick test, and if all went well, you will see that resizing the card window includes positioning the first four buttons using the resizeStack handler that we added, and it’s positioning of the second set of four buttons using the Geometry Manager. With the settings we used, the results should be much the same, except that the second set of four buttons will remain a fixed distance away from the bottom of the card window.

There is a lot of power in the Geometry Manager and you should take a look at the other abilities it has at the reference link shown at the end of this chapter. However, it is not the best way to deal with mobile screen sizes.

**Resolution independence**

LiveCode 6.5, and beyond...

One of the features listed in *Chapter 1, LiveCode Fundamentals*, has already been developed and is present in LiveCode 6.5. The new feature is Resolution Independence.

Now, after seeing the two complex ways of adjusting the screen size, you may be able to forget all of it. In LiveCode 6.5, a new feature called Resolution Independence was introduced, and the correct use of this feature will make many of the layout difficulties go away.
There are two approaches in place to help you deal with different device sizes and aspect ratios: the Multiple Density Support and the Full Screen Scaling Mode. Some of the concepts are a little tricky, but a few screenshots will hopefully make them clear!

**Multiple density support**

So far in this chapter, we overlooked the issue of the DPI (dots per inch) of devices. For Android devices, there are hundreds of different resolutions and DPI values. With iOS, there are only a few variations of DPI and resolution. The simplest these problem cases to examine, is where you want the same app to look like and work in the same manner as iPhone 3GS and iPhone 4 Retina or later versions of iPhones. Both have the same size screen, but iPhone 4’s retina display has twice the DPI. This URL shows you the difference between all the current iPhones: [http://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions](http://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions).

With the MobGUI and Geometry Manager solution, you are effectively storing layout values or instructions behind the scenes that are ready to adapt to the user’s device screen size. The Multiple Density Support in LiveCode 6.5 is an easier way to solve this issue.

**Pixels and points**

A pixel is an illuminated dot in an image or of a screen, whereas a point is a unit of measurement in print, generally, 1/72nd of an inch. If a screen has exactly 72 DPI, the two would be the same, and in the earlier days of Macintosh computers, this was pretty much the case. More modern Macs and PCs and most mobile devices are a lot more detailed than 72 DPI. With LiveCode 6.5, and its later versions, you can now make a stack that works in points, that will then fill the device screen by taking the image of each card and by applying a scaling factor to it.

While doing such scaling, something must be assumed to be of natural size. For iOS, LiveCode uses non-Retina screens as a scale factor of 1X and Retina screens are a scale factor of 2X. iPhone 5 and 6 introduced more sizes. A scale factor of 3X is required for iPhone 6 Plus. With Android, things are more complicated as it has at least seven levels of scaling factor.

In fact, really, there are an infinite number of scaling factors going on behind the scenes. However, you generally don’t have to worry about them because LiveCode automatically gives you the ability to show different versions of an image depending on the DPI of the device in question.
Image naming convention

LiveCode will look at the file names of the included images (that have been added in the Copy Files part of the Standalone Application Settings dialog) and will load in the appropriately named version of the image. In the example shown here, two images have been added, icon.png and icon@2x.png. To make it easier to spot which one you are seeing, 512 and 1024 have been added to what would otherwise be two resolutions of the same icon:

![512 and 1024 versions of an icon](image)

This image is slightly magnified here to help you see that the 512 version of the image is lower resolution than the 1024 version.

The names used for iOS are:

- `imagename.ext` (for example, `flowers.png`)
- `imagename@2x.ext` (for example, `flowers@2x.png`)
- `imagename@3x.ext` (for example, `flowers@3x.png`) for iPhone 6 Plus

The first one is considered as the 1X scaled version and the @2x name is considered as the 2X scaled version and the same goes for the @3x version.

Note that you also need to specify the appropriate Splash Screen files in the LiveCode Standalone Application Settings. Otherwise, the iPhone will use lower resolution images. This can be verified by examining the screenRect of the device. Even though iPhone 6 Plus has a physical resolution of 1080 x 1920, use the 3x image sizes of 1242 x 2208. The phone automatically downsizes the image. 2x images will be displayed fine on iPhone 6.
The names and scaling factors used for Android are more varied:

- imagename@ultra-low.ext - 0.25X
- imagename_extra-low.ext - 0.5X
- imagename low.ext - 0.75
- imagename medium.ext - 1X
- imagename high.ext - 1.5X
- imagename@extra-high.ext - 2x
- imagename@ultra-high.ext - 4x

In actual practice, you may find that you only need two or three versions of an image to make an image look good enough on a wide range of DPI devices.

**The full-screen scaling mode**

Prior to LiveCode 6.5 and its multiple density support, you had to do all the hard work by yourself! At least now you can design for just iPhone or iPad and have Retina versions of these devices looked after by LiveCode. With Android though, you still need to have a variety of layouts or alternately, have code that would position the interface elements based on the aspect ratio and pixel dimensions of the device.

With the density support, you only have to take care of the aspect ratio, but there are still a lot of these to be taken care of. For utility-like applications, you will most likely have to go to this trouble, so that users get the expected experience. However, there are many types of applications that are more graphical and that can take advantage of the full screen scaling modes: empty, exactFit, showAll, noBorder, and noScale. Let's take a look at what these are...

**Syntax**

The general syntax, by the way, is to type this in your stack script:

```
on preopenstack
  set the fullscreenmode of the current stack to "showAll"
end preopenstack
```

The quotes are required for the four active modes. The empty mode does not need quotes.
Chapter 3

The empty mode

If `fullscreenmode` is `empty`, the card area is changed to match the device screen. That is, the top left card is in the top-left corner of the device screen and the card width and height match the actual pixel width and height of the device. Well, except if you have taken advantage of the multiple density support discussed earlier! Essentially, empty is the existing behavior in the earlier versions of LiveCode.

The showAll mode

This mode arrived slightly later than the other modes and is present in LiveCode 6.6 and its later versions. With `showAll`, as the `fullscreenmode`, the card’s contents will be scaled, so that all of it is within the device’s screen. For devices that have a different aspect ratio than your card, elements that were outside of the card area are revealed. Suppose you create a graphical book app where you want the full height of the card to exactly fill the height of the device screen, but you don’t want black bars down the sides on a wider device, then you could extend the background pattern beyond the left- and right-hand side of the card area. On a narrower device, such as iPad, you would see the 4:3 area of the card. On a wider device, say iPhone 5, the extra background would be revealed. The following illustration shows which area of the background image will be seen on different devices:
The letterbox mode

The letterbox mode is identical to showAll except that the areas beyond the card area are hidden and leave you with the typical movie letterbox effect, hence the name!

The noBorder mode

The noBorder mode is very useful, but takes some time to get used to. Let’s say you are making a graphical adventure game, along the lines of Myst, and you want to use the same graphics for all devices. You could create a scene that has an aspect ratio of 14:9 and make sure that the important content is not too close to the edges. When the scene is viewed in the noBorder mode on an iPad, you would see the full height of the scene and most of the width. When viewed on iPhone 5, you would see the full width of the scene and most of the height. As the name suggests, the card area would be scaled, so that there are no borders, as there would be in the letterbox mode. The following illustration shows which areas of an original 14:9 photograph would be visible for a 4:3 iPad and a 16:9 iPhone:
The **exactFit** mode

As the name implies, **exactFit** would take the card’s content and squish it to fill the device’s screen. It’s hard to think of a use case for this mode, but perhaps, if you perform some sort of artistic visualizer, the squishing wouldn’t matter much!

As mentioned previously, the Geometry Manager has a lot of powerful features. If your interest is also in desktop applications, take a look at the lesson at:

http://lessons.runrev.com/s/lessons/m/4067/l/19026-Geometry-Manager

**Summary**

The trick with easy-to-use tools, such as LiveCode, is to create mobile apps that users think were created with the hard-to-use native tools, such as Xcode. You can achieve this because of LiveCode’s ability to call upon native features and because you can make interfaces that look appropriate.

In this chapter, we covered several ways to achieve this goal by calling native OS features using simple LiveCode commands. We prepared images to be used for button states and made buttons that look OS-specific by adding these images. We also created controls that look like iOS native controls with MobGUI and laid out the interface with code, with the Geometry Manager.

So far, these have all been small test stacks to get us warmed up! Next, we’re going to look at the general application structure to make a fully fleshed out utility application.
Where to buy this book


Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.