Quick answers to common problems

LiveCode Mobile Development Cookbook

90 practical recipes for creating cross-platform mobile applications with the power of LiveCode

Dr Edward Lavieri

[PACKT] PUBLISHING

LiveCode Mobile Development Cookbook

90 practical recipes for creating cross-platform mobile applications with the power of LiveCode

Dr Edward Lavieri

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

LiveCode Mobile Development Cookbook

First published: September 2014

Production reference: 1050914

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78355-882-7

www.packtpub.com

Cover image by Neha Rajappan (neha.rajappan1@gmail.com)

Credits

Author
Dr Edward Lavieri

Reviewers
Erik Beugelaar
Guanhua Chen
Cecil Costa
Theo Heselmans
Simon Sunatori

Commissioning Editor
Saleem Ahmed

Acquisition Editor
Sam Wood

Content Development Editor
Arvind Koul

Technical Editors
Veronica Fernandes
Anand Singh

Copy Editors
Aditya Nair
Stuti Srivastava

Project Coordinator
Priyanka Goel

Proofreaders
Ting Baker
Maria Gould
Joel T. Johnson

Indexer
Tejal Soni

Production Coordinator
Shantanu Zagade

Cover Work
Shantanu Zagade

About the Author

Dr Edward Lavieri is a veteran game designer and developer with a strong academic background. He earned a Doctorate of Computer Science from Colorado Technical University and three Masters of Science degrees in Management Information Systems (Bowie State University), Education in Instructional Design (Capella University), and Operations Management (University of Arkansas), thus demonstrating his passion for academic pursuits. He has been developing and teaching computer-related courses since 2002. He retired from the U.S. Navy after 25 years as an Intelligence specialist and Command master chief.

As the founder and creative director of three19, which is a software design and development studio, Edward is constantly developing software. He uses LiveCode as one of his primary prototyping and development tools. He focuses on developing adaptive learning systems, educational games, and mobile apps.

He has authored *LiveCode Mobile Development Hotshot*, *Packt Publishing*; *Software Consulting: A Revolutionary Approach*, *CreateSpace*; and was the technical editor of *Excel Formulas and Functions for Dummies*, *John Wiley & Sons, Inc*. He has also authored numerous computer science and information systems college courses.

My deepest appreciation goes to Brenda, my loving wife and partner in life. Without her support, this book would not have been possible. I would also like to thank the expert team at Packt Publishing including Arvind Koul, Sam Wood, Sheetal Sarkar, the reviewers, the technical editors, the proofreaders, the indexer, and the marketing team. It is a pleasure to have worked with such an efficient team.

About the Reviewers

Erik Beugelaar is a programmer who is focused on visual developer tools, giving a software designer the feeling that making software is an art because of the way the graphical user interface is built using the visual toolset.

After trying many visual developer tools during the last 10 years, LiveCode got his special attention in 2010. It was the natural language syntax that could express the way things could be done, and more specifically, the writing of English-like sentences had become an art too.

Nowadays, he is using LiveCode to examine and analyze data for financial purposes.
As a developer of tools for other developers, he is looking forward to the Next Generation version of LiveCode.

Along with his fulltime professional career, he is also actively involved in voluntary projects in Kenya. Whenever possible, he promotes the educational value of using LiveCode in classes.

Guanhua Chen is a doctoral student at the University of Miami, majoring in Teaching and Learning (specialization in Science, Technology, Engineering, and Mathematics). He gained his Master's degree from the Learning, Design, and Technology program at the University of Georgia. His research is focused on the intersection between technology and STEM education. He is especially interested in utilizing various technologies as cognitive tools as well as innovative assessment instruments in science education.

Cecil Costa is a freelance developer and founder of Conglomo Limited (`www.conglomo.es`), which offers development and training programs. In his professional career, he has created projects by himself and has also worked for a variety of companies from small to large ones, such as IBM, Qualcomm, Spanish Lottery, and DIA%.

He develops in a variety of computer languages (such as C++, Java, Objective-C, JavaScript, Python, and so on) in different environments (iOS, Android, Web, Mac OS X, Linux, Unity, and so on) because he thinks that a good developer needs to learn every kind of programming language in order to open his mind, and only then will he really know what development is.

He has worked with LiveCode, creating educational music games for Acción Piano School in Spain. You can view some of his work at `www.smartboardmusic.org`.

I would like to thank Victoria López Messeguer for giving me the opportunity to learn and use LiveCode.

Theo Heselmans is an IBM Champion for Collaboration Solutions. In 1993, he started working as a Notes/Domino consultant with Version 3. He delighted (and still does) many customers with custom applications (complex CRM systems, web content management solutions, project and document management, reporting, and so on).

In 2001, he founded his own company, Xceed (`www.xceed.be`), which is now a proud member of the Penumbra Group.

Since 2009, he has been responsible for Engage (also known as BLUG). During their events, he gives users, speakers, IBMers, and business partners the opportunity to share information, collaborate, and network. Theo visits other LUGs regularly and has been to 19 Connect/Lotusphere conferences.

He is an avid believer in all things *social*.

He loves to talk about Notes, mobile development (preferably with LiveCode), user experience, Excel, and iPad (and wine)! You can follow him at `@theoheselmans`.

Simon Sunatori, P.Eng./ing., M.Eng. (Engineering Physics), F.N.A., IEEE-SM, WFS-LM, is a computer programmer who has written Unix and IBM mainframe software programs, such as microelectronics device characterization, hypertext search facility, electronic democracy and voting groupware, idiot-proof push-button automation, integrated version control and configuration management, and intelligent workflow automation systems. He also wrote Apple Macintosh applications such as *HyperInfo Intelligent Knowledge Object Organisation System*, which organizes and processes information as knowledge objects and *Multi-Lingual Food Nutrition Knowledge Matrix*, which organizes hundreds of foods and nutrients in a simple and consistent manner in both English and French. He has written CGI scripts such as *Electronic Commerce Transaction Processing*, *Multi-Lingual Presentation from a Single Database*, and *Printable Calendar Generator*. He has reported more than 2,000 bugs for Apple Mac OS X and more than 1,000 bugs for RunRev LiveCode. He wrote a script to generate both a biography and an obituary from a single source with the push of a button, using various text-processing techniques in LiveCode. His biography and obituary can be found at `http://www.hyperinfo.ca/GS.Sunatori/HomePage_Biography.html` and `http://www.hyperinfo.ca/GS.Sunatori/HomePage_Obituary.html`, respectively.

Table of Contents

Preface

LiveCode is a powerful programming environment with an easy-to-use scripting language. As a development environment, LiveCode has the capability to publish apps for both Android and iOS mobile devices. This, coupled with its ability to publish to desktop computers, makes LiveCode a multiplatform development environment.

The number of available mobile development tools continues to increase. Despite the growing number of options, LiveCode's English-like programming language makes mobile app development a more efficient process than that of other development tools. The speed at which we can prototype and provide final apps with LiveCode is impressive.

You'll find this book's recipes a great tribute to LiveCode and, hopefully, a useful reference for you.

What this book covers

Chapter 1, *LiveCode Mobile Basics*, introduces you to the concepts and steps that are required to set up your computer to develop Android and iOS applications using LiveCode. You'll learn how to set up your development environments, what icons and images are required for the apps you create, how to use the mobile simulator, how to run your apps on physical devices, and how to get your apps into the global marketplace.

Chapter 2, *Human-computer Interfaces*, demonstrates a variety of human-computer interface objects and how they can be used in your mobile apps. In addition to learning about buttons, cards, input boxes, dialog windows, and geometric shapes, you'll explore password masking, card navigation, and how to change an object's properties.

Chapter 3, *Loops and Timers*, has you examine the LiveCode scripting required to implement and control count-up and countdown timers. You'll also learn how to use loops to count and iterate through a list.

Chapter 4, *Managing Text*, offers you a complete coverage of how to manage text to include reading the user input, searching, replacing, combining, encrypting, writing, reading, sorting, formatting, and appending. As a bonus example, this chapter includes the LiveCode scripting required to create the Pig Latin text.

Chapter 5, *Communications*, shows you how to initiate a phone call from within your mobile apps. You will also learn how to format and send an e-mail.

Chapter 6, *Data Structures*, explores how to use advance data structures to include arrays (both one- and multidimensional), XML, SQLite, and MySQL.

Chapter 7, *External Media*, covers everything you need to know about external media for your mobile apps. You'll learn how to load an external image, how to capture an image from a mobile device's camera, how to resize an image, and how to play movie and audio clips.

Chapter 8, *Using MobGUI*, provides an introduction to using the MobGUI LiveCode plugin to accelerate your mobile app development. This chapter provides hands-on recipes using all of MobGUI's interface objects.

Chapter 9, *Using Animation Engine*, presents 10 recipes that explore the power of the Animation Engine plugin's capabilities. You'll learn how to move objects, stop objects, change the speed of objects, simulate gravity, and more.

Chapter 10, *Miscellaneous*, introduces you to LiveCode's mathematical computation abilities. In addition to gaining hands-on experience with math operations, you'll learn how to open and query a web page, how to use the geometry manager, how to use invisible objects, how to detect the user's mobile operating system, and how to take snapshots of cards and specific areas on a card.

What you need for this book

To complete the recipes in this book, you will need to download and install a current license of LiveCode. You can use either a community or professional license version 6.x or higher.

In order to develop mobile apps for Android devices, you'll need to download and install the latest Android SDK. If you are using a Windows computer, you'll also need to have the Java SDK.

If you are developing iOS mobile apps, you'll need to download the Apple iOS SDK and Xcode. Both of these tools are available to you as an Apple developer.

Configuring your computer for use with LiveCode is pretty straightforward when you are developing for iOS devices. This is true for Windows, Mac, and Linux machines. When developing for Android devices, the configuration process can be a bit more difficult on a PC running Windows than on a Mac. Please consult the LiveCode documentation if you run into any problems.

To complete the recipes in *Chapter 8*, *Using MobGUI*, you'll need to have a copy of the MobGUI LiveCode plugin. For *Chapter 9*, *Using Animation Engine*, you'll need a copy of the Animation Engine plugin.

Chapter 1, *LiveCode Mobile Basics*, will help you download and install these software tools on your development computer.

LiveCode is available for Mac OS, Windows, and Linux based computers. The user interface for each version is slightly different. The examples and images in this book are from Mac OS.

Who this book is for

This book is written for anyone who wants to get started with developing mobile apps using LiveCode. Some knowledge of the LiveCode integrated development environment (IDE) and scripting language is assumed, although the recipes were written in a manner that allows pure novices, as well as experienced developers, to follow the steps. No experience with developing mobile apps is required to start using this book.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Create a text entry field and name it `fldUsername`."

A block of code is set as follows:

```
go to card "Main"
go to card 3
go to next card
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
answer quote & englishWord & quote & " is " &quote & pigLatinWord & quote\
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Select **Mobile Support** from the left navigation pane of the **Preferences** window."

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
LiveCode Mobile Basics

In this chapter, we will cover the following recipes:

- Setting up your mobile environment for iOS development
- Setting up your mobile environment for Android development
- Defining icons and images for iOS development
- Defining icons and images for Android development
- Configuring standalone application settings for iOS applications
- Configuring standalone application settings for Android applications
- Using the simulator
- Saving a standalone mobile app

Introduction

In this chapter, you will learn how to accomplish tasks related to working with iOS and Android apps and their respective mobile application marketplaces. You will learn how to set up your development environments, what icons and images are required for apps, how to use the simulator, how to run the apps on your devices, and how to get your apps in the global marketplace.

Setting up your mobile environment for iOS development

LiveCode enables us to create mobile applications for iOS devices such as the iPad, iPhone, and iPod Touch. Before we can start developing for these devices, we need a few things from Apple, which is the creator of iOS. This recipe details how to accomplish this.

Getting ready

Before you can complete this recipe, you will need to be a certified Apple Developer and have your account information (username and password) available. See `https://developer.apple.com` for more information.

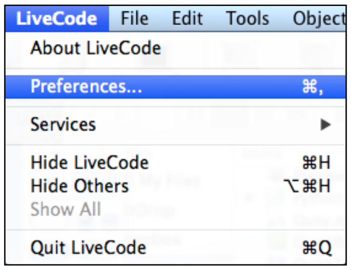
How to do it...

Once you have your Apple Developer account and LiveCode installed on your development computer, you are ready to get started with this recipe.

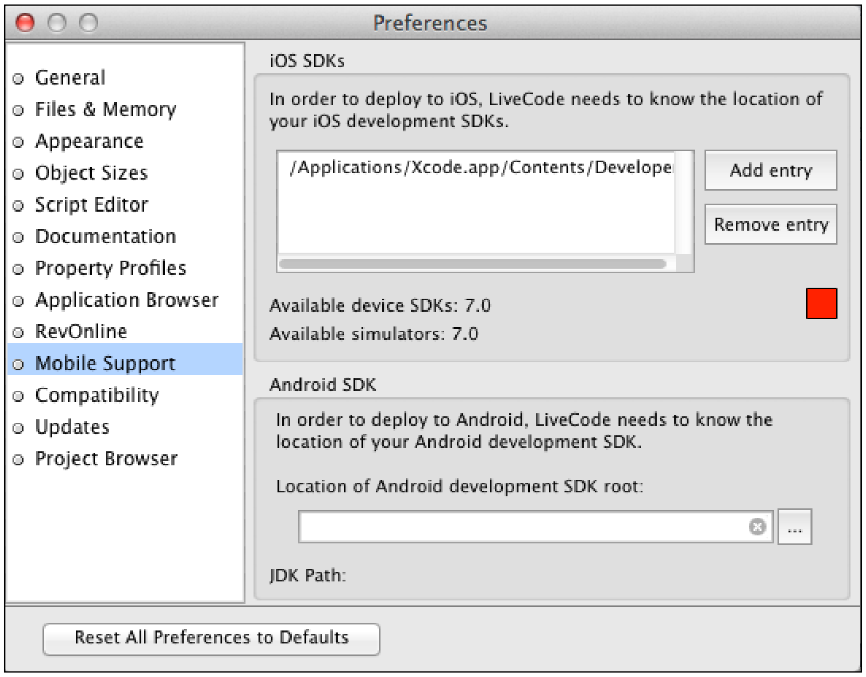
1. Log in to your Apple Developer account.
2. Download the iOS SDK.
3. Download and install Xcode.

> Xcode downloads are several gigabytes in size and can take a long time to download depending upon your Internet connection speed. Do not worry if the download takes longer than you anticipated.

4. Select **Preferences...** from the LiveCode drop-down menu.



5. Select **Mobile Support** from the left navigation pane of the **Preferences** window.



iOS SDKs

In order to deploy to iOS, LiveCode needs to know the location of your iOS development SDKs.

/Applications/Xcode.app/Contents/Develope

Available device SDKs: 7.0
Available simulators: 7.0

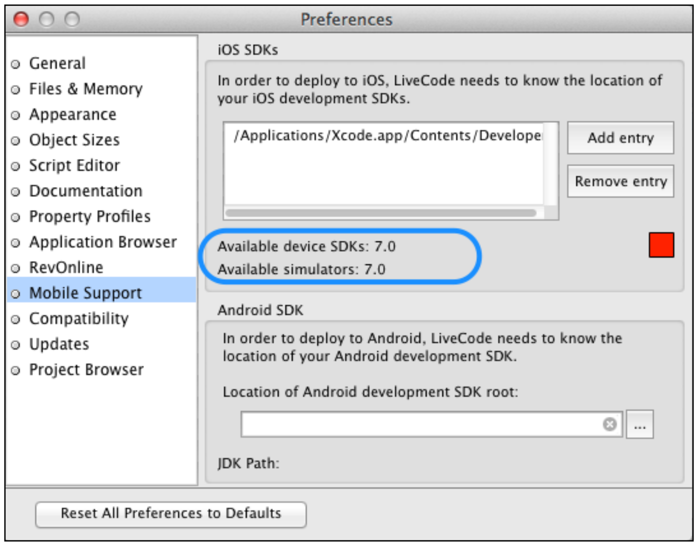
Android SDK

In order to deploy to Android, LiveCode needs to know the location of your Android development SDK.

Location of Android development SDK root:

JDK Path:

6. Use the **Add entry** button to navigate to the location of Xcode on your development computer. When this is done correctly, you will see version numbers listed under **Available device SDKs** and **Available simulators**.



iOS SDKs

In order to deploy to iOS, LiveCode needs to know the location of your iOS development SDKs.

/Applications/Xcode.app/Contents/Develope

Available device SDKs: 7.0
Available simulators: 7.0

Android SDK

In order to deploy to Android, LiveCode needs to know the location of your Android development SDK.

Location of Android development SDK root:

JDK Path:

How it works...

In this recipe, we used our Apple Developer account to download and install the latest iOS SDK and Xcode software. Next, we configured LiveCode so that LiveCode's IDE is linked with our iOS development SDK. This will now allow us to develop iOS applications with LiveCode.

There's more...

It is important to ensure that you do not have any conflicts with your computer's operating system, version of the SDK, or Xcode. For example, if you are running Mountain Lion or Mavericks on your Mac development computer, it is recommended that you use Xcode 5.0.2 and SDK 7.0. Consult the latest LiveCode release documentation for updated information.

See also

- The *Setting up your mobile environment for Android development* recipe
- The *Configuring standalone application settings for iOS applications* recipe

Setting up your mobile environment for Android development

There are a lot of devices made by a multitude of mobile hardware devices that run the Android operating system. In order for us to develop for Android devices, we must have the Android SDK installed on our development computer. In addition, we must configure LiveCode so that it knows where the SDK is installed.

Getting ready

Unlike developing for iOS, you do not need a developer account to obtain the Android SDK.

How to do it...

Setting up your development environment so that you can develop Android apps using LiveCode is accomplished by the following steps:

1. Download the latest Android SDK from the following site:

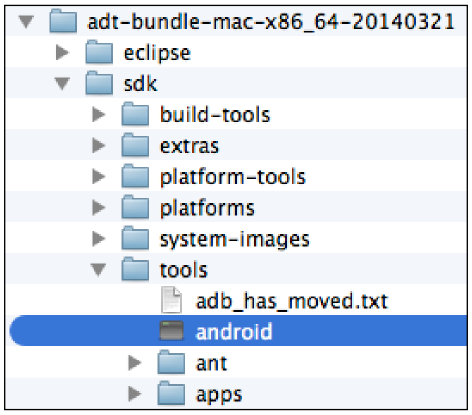
   `http://developer.android.com/sdk/index.html`

   > You should only have to download the Android SDK once. The SDK Manager gives you the flexibility to install additional packages as well as future updates.

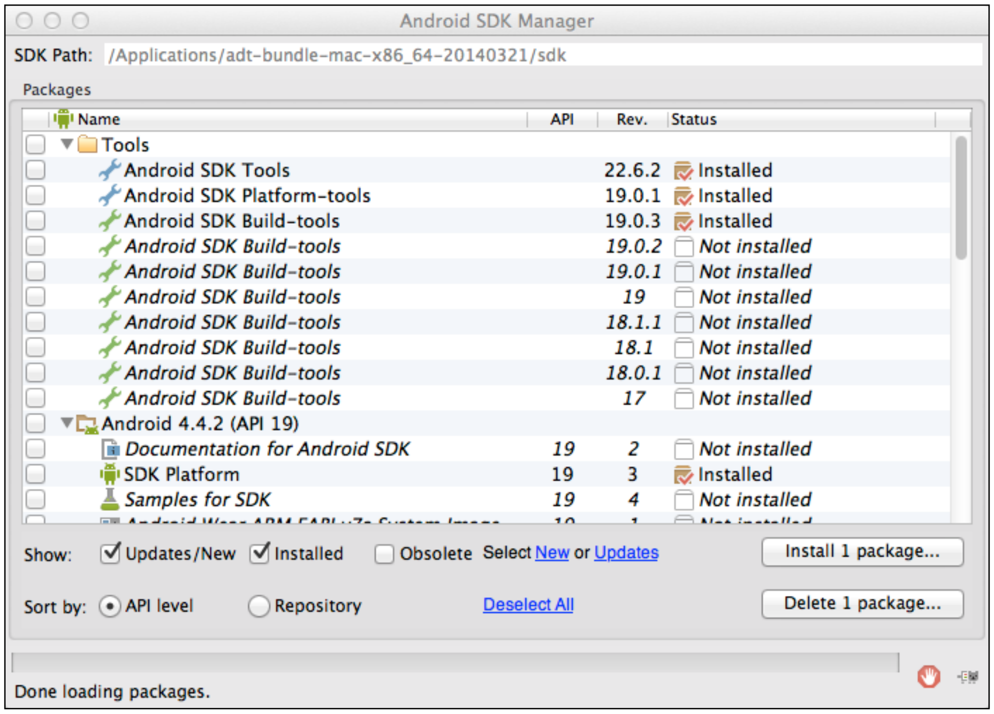
2. If you are using a Windows or Linux based computer to develop your Android app, you will also need to download and install the Java SDK from the following site:

   `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

3. Double-click on the Android SDK compressed file (it will be named similar to `adt-bundle-mac-x86_64-20140321.zip`) to uncompress/unzip the package.
4. Install the Android SDK and, if you are using a PC, the Java SDK. On Mac, navigate to the newly installed Android SDK folder. Run the **android** program by navigating to **sdk** | **tools**. This loads the Android SDK Manager.



5. Using the Android SDK Manager, ensure that you have the desired tools and documentation installed. Most developers do fine with the defaults; your situation might be different. Also, you should install any available updates.



Android SDK Manager

SDK Path: /Applications/adt-bundle-mac-x86_64-20140321/sdk

Packages

Name	API	Rev.	Status
☐ Tools			
☐ Android SDK Tools		22.6.2	Installed
☐ Android SDK Platform-tools		19.0.1	Installed
☐ Android SDK Build-tools		19.0.3	Installed
☐ *Android SDK Build-tools*		*19.0.2*	*Not installed*
☐ *Android SDK Build-tools*		*19.0.1*	*Not installed*
☐ *Android SDK Build-tools*		*19*	*Not installed*
☐ *Android SDK Build-tools*		*18.1.1*	*Not installed*
☐ *Android SDK Build-tools*		*18.1*	*Not installed*
☐ *Android SDK Build-tools*		*18.0.1*	*Not installed*
☐ *Android SDK Build-tools*		*17*	*Not installed*
☐ Android 4.4.2 (API 19)			
☐ *Documentation for Android SDK*	*19*	*2*	*Not installed*
☐ SDK Platform	19	3	Installed
☐ *Samples for SDK*	*19*	*4*	*Not installed*

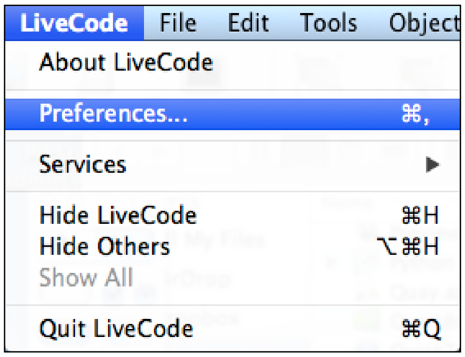
Show: ☑ Updates/New ☑ Installed ☐ Obsolete Select New or Updates — Install 1 package...

Sort by: ● API level ○ Repository — Deselect All — Delete 1 package...

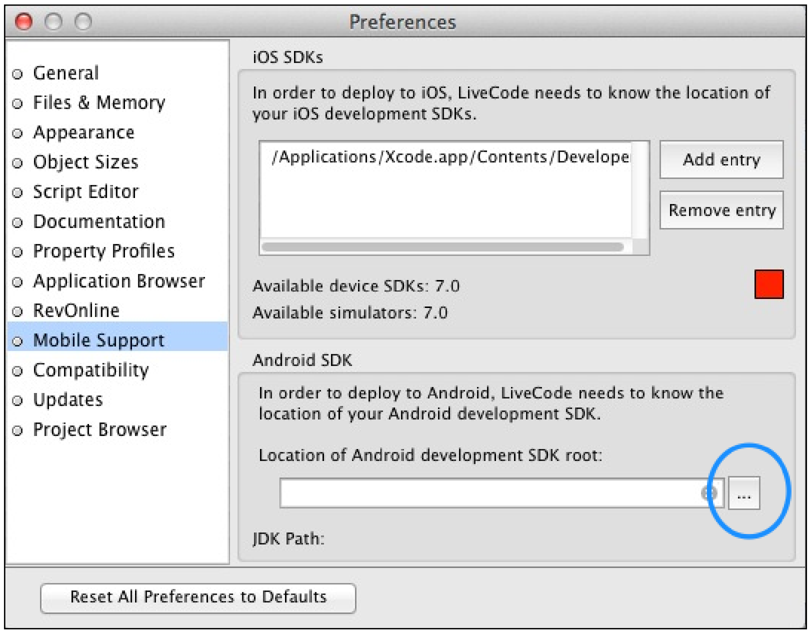
Done loading packages.

6. Ensure that you install the Android 2.2 (API 8) SDK platform tools. If you fail to do this, you will run into an error when trying to configure LiveCode for Android development. The Android SDK Manager will handle downloading, unzipping, and installing the tools and updates you select.

7. Select **Preferences...** from the LiveCode drop-down menu.



8. Select **Mobile Support** from the left navigation pane of the **Preferences** window.



iOS SDKs

In order to deploy to iOS, LiveCode needs to know the location of your iOS development SDKs.

/Applications/Xcode.app/Contents/Develope

Available device SDKs: 7.0
Available simulators: 7.0

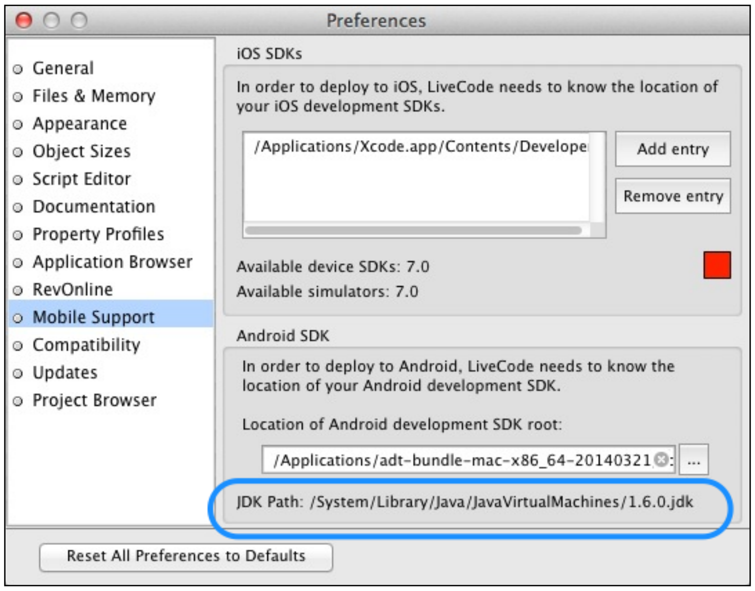
Android SDK

In order to deploy to Android, LiveCode needs to know the location of your Android development SDK.

Location of Android development SDK root:

JDK Path:

9. Use the **...** button to navigate to the location of the Android SDK on your development computer. When this is done correctly, you will see your specific path listed after **JDK Path** on the **Preferences** dialog window.



iOS SDKs

In order to deploy to iOS, LiveCode needs to know the location of your iOS development SDKs.

/Applications/Xcode.app/Contents/Develope

Available device SDKs: 7.0
Available simulators: 7.0

Android SDK

In order to deploy to Android, LiveCode needs to know the location of your Android development SDK.

Location of Android development SDK root:

/Applications/adt-bundle-mac-x86_64-20140321

JDK Path: /System/Library/Java/JavaVirtualMachines/1.6.0.jdk

How it works...

In this recipe, we downloaded and installed the latest Android SDK. Next, we configured LiveCode so that LiveCode's IDE is linked to our Android development SDK. This will now allow you to develop Android applications with LiveCode.

There's more...

If you run into any installation problems, as with any online software, it is a good idea to check the software documentation instructions. Typically, there is a `readme` text file in the directory with the installation files.

Configuring your computer for use with LiveCode is pretty straightforward when you are developing for iOS devices. This is true for Windows, Mac, and Linux machines. When developing for Android devices, the configuration process can be a bit more difficult on a PC running Windows than on a Mac. Please consult the LiveCode documentation if you run into any problems.

See also

- The *Setting up your mobile environment for iOS development* recipe
- The *Configuring standalone application settings for Android applications* recipe

Defining icons and images for iOS development

When we develop apps for iOS devices, we must provide specifically formatted icon and splash screen images. Because of the different orientations (portrait and landscape) and screen sizes, several different versions of each image (icon and splash screen) are required.

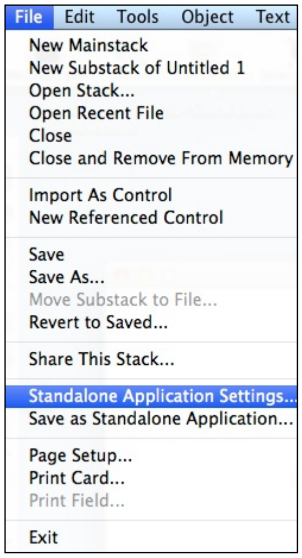
Getting ready

You will require software to create the original graphics as well as to export them in the proper sizes.

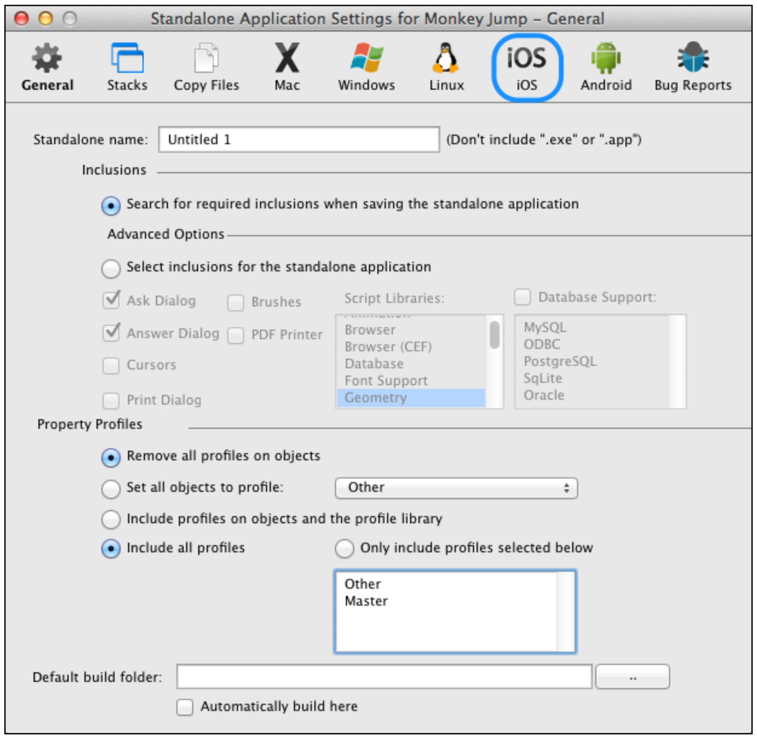
How to do it...

The icons and images for iOS development are defined using the following steps:

1. Select the **Standalone Application Settings...** option from the **File** drop-down menu.



2. Click on the iOS icon in the top row of the Standalone Application Settings dialog box.



Standalone name: Untitled 1 (Don't include ".exe" or ".app")

Inclusions

[✓ Search for required inclusions when saving the standalone application]

Advanced Options

Select inclusions for the standalone application

[✓ Ask Dialog] Brushes Script Libraries: Database Support:

[✓ Answer Dialog] PDF Printer

Cursors

Print Dialog

Browser, Browser (CEF), Database, Font Support, Geometry

MySQL, ODBC, PostgreSQL, SqLite, Oracle

Property Profiles

[✓ Remove all profiles on objects]

Set all objects to profile: Other

Include profiles on objects and the profile library

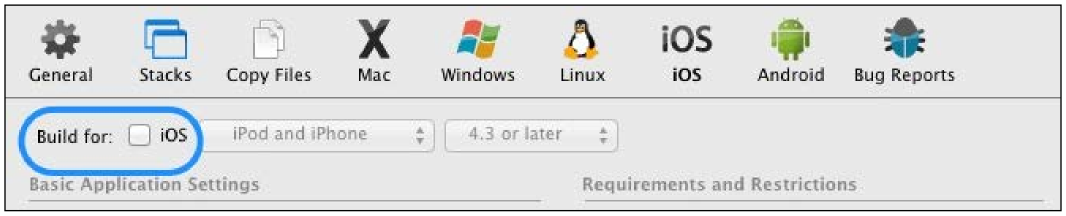
[✓ Include all profiles] Only include profiles selected below

Other
Master

Default build folder:

Automatically build here

3. Ensure that the checkbox next to the iOS build option is checked. Once you select this option, a checkmark will appear in the box and the remaining options will be editable. Until you select this option, all options are disabled. Also, once you indicate that you will be saving mobile versions, any selected desktop deployment options will be deselected automatically.

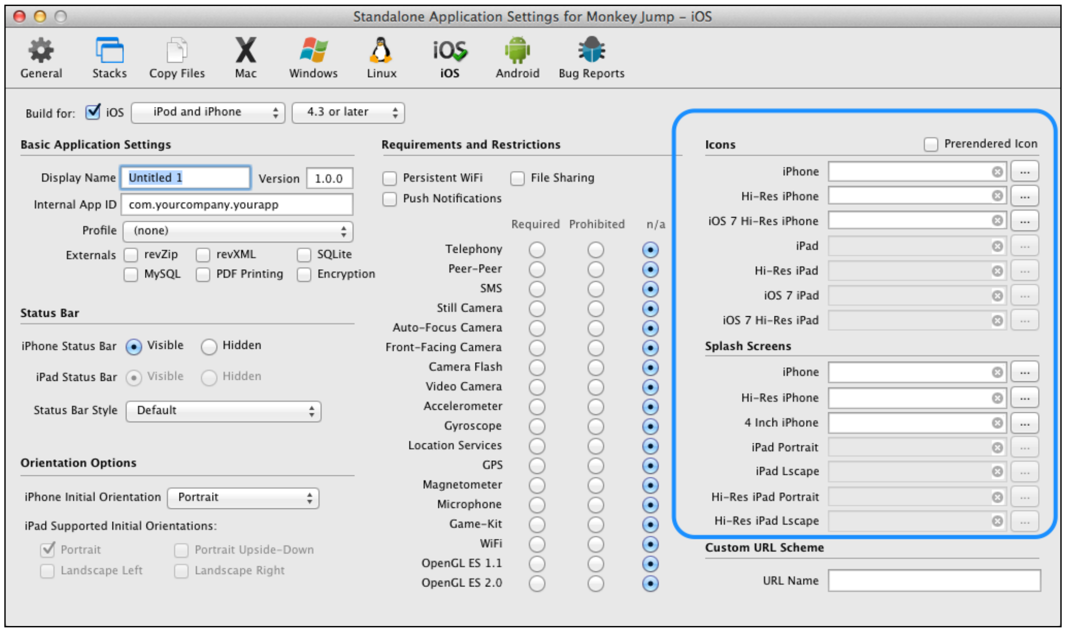


Build for: ☐ iOS | iPod and iPhone | 4.3 or later

Basic Application Settings

Requirements and Restrictions

4. Use the **...** buttons to the right of each icon / splash screen entry to upload your images.



Build for: ☑ iOS | iPod and iPhone | 4.3 or later

Basic Application Settings

Display Name: Untitled 1 | Version: 1.0.0

Internal App ID: com.yourcompany.yourapp

Profile: (none)

Externals: ☐ revZip ☐ revXML ☐ SQLite ☐ MySQL ☐ PDF Printing ☐ Encryption

Status Bar

iPhone Status Bar: [✓ Visible] Hidden

iPad Status Bar: [✓ Visible] Hidden

Status Bar Style: Default

Orientation Options

iPhone Initial Orientation: Portrait

iPad Supported Initial Orientations: ☑ Portrait ☐ Portrait Upside-Down ☐ Landscape Left ☐ Landscape Right

Requirements and Restrictions

☐ Persistent WiFi ☐ File Sharing ☐ Push Notifications

	Required	Prohibited	n/a
Telephony	○	○	●
Peer-Peer	○	○	●
SMS	○	○	●
Still Camera	○	○	●
Auto-Focus Camera	○	○	●
Front-Facing Camera	○	○	●
Camera Flash	○	○	●
Video Camera	○	○	●
Accelerometer	○	○	●
Gyroscope	○	○	●
Location Services	○	○	●
GPS	○	○	●
Magnetometer	○	○	●
Microphone	○	○	●
Game-Kit	○	○	●
WiFi	○	○	●
OpenGL ES 1.1	○	○	●
OpenGL ES 2.0	○	○	●

Icons ☐ Prerendered Icon

iPhone ...
Hi-Res iPhone ...
iOS 7 Hi-Res iPhone ...
iPad ...
Hi-Res iPad ...
iOS 7 iPad ...
iOS 7 Hi-Res iPad ...

Splash Screens

iPhone ...
Hi-Res iPhone ...
4 Inch iPhone ...
iPad Portrait ...
iPad Lscape ...
Hi-Res iPad Portrait ...
Hi-Res iPad Lscape ...

Custom URL Scheme

URL Name

5. Ensure that your icon images have been exported with the following pixel dimensions:

Icon	Image size
iPhone	57 x 57
Hi-Res iPhone	114 x 114
iOS 7 Hi-Res iPhone	120 x 120
iPad	72 x 72
Hi-Res iPad	144 x 144
iOS 7 iPad	76 x 76
iOS 7 Hi-Res iPad	156 x 156

6. Ensure that your splash screen images have been exported with the following dimensions:

Splash screen	Image size
iPhone	320 x 480
Hi-Res iPhone	640 x 960
4 Inch iPhone	640 x 1136
iPad Portrait	768 x 1024
iPad Landscape	1024 x 768
Hi-Res iPad Portrait	1536 x 2048
Hi-Res iPad Landscape	2048 x 1536

How it works...

Pointing the file-selection window to each specific icon and splash screen is easy work. If you attempt to upload an image that does not have the correct dimensions, LiveCode will present you with an error message. The files you select are embedded in your app's binary and are used when uploading it to the App Store.

There's more...

If you only enable one orientation, you will not be required to upload images to support the other orientation. For example, if your mobile app only supports the portrait orientation, you do not need to upload landscape splash screens.

See also

- The *Setting up your mobile environment for iOS development* recipe
- The *Defining icons and images for Android development* recipe

Defining icons and images for Android development

When we develop apps for Android devices, we must provide specifically formatted icon and splash screen images. Because of the different orientations (portrait and landscape) and screen sizes, several different versions of each image (icon and splash screen) are required.

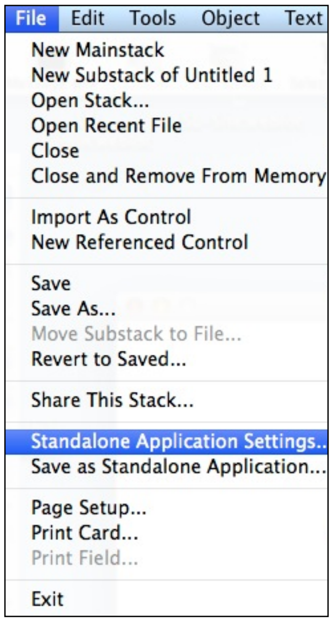
Getting ready

External graphic creation and editing software is required to create the original graphics as well as to export them in the proper sizes.

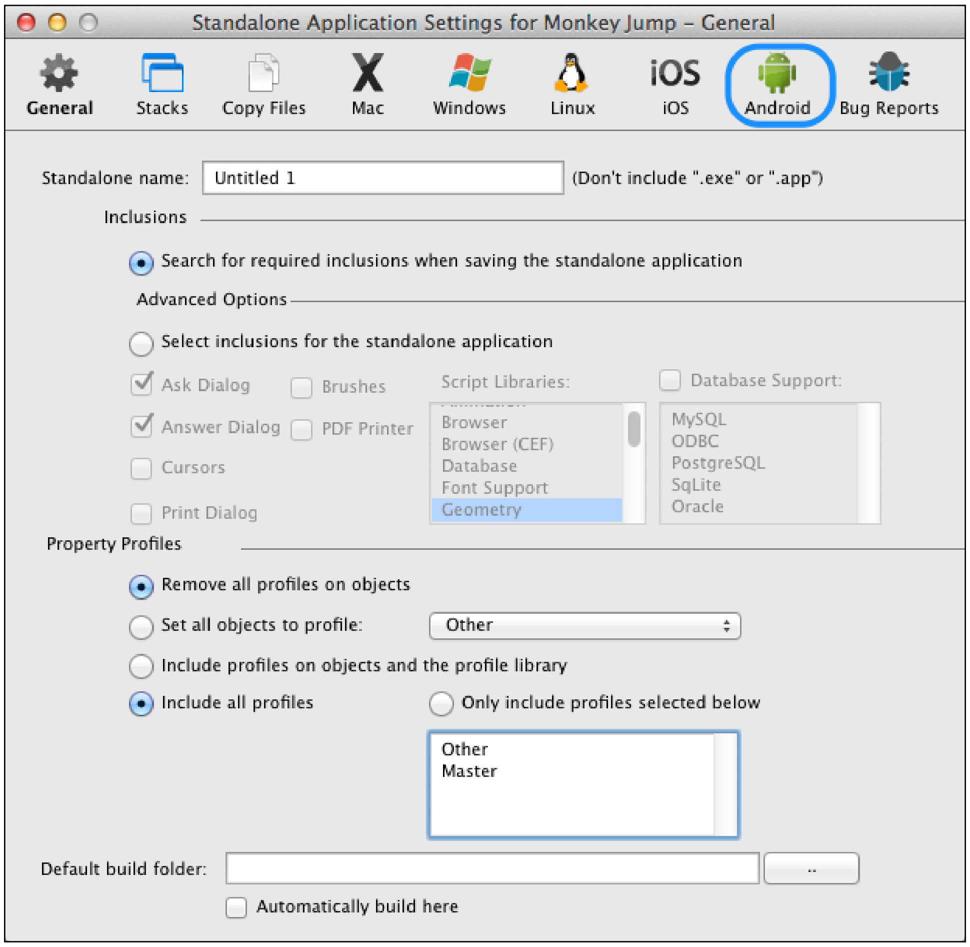
How to do it...

The icons and images for Android development are defined using the following steps:

1. Select the **Standalone Application Settings...** option from the **File** drop-down menu.



2. Click on the icon on the top row of the Standalone Application Settings dialog box.



Standalone name: Untitled 1 (Don't include ".exe" or ".app")

Inclusions

● Search for required inclusions when saving the standalone application

Advanced Options

○ Select inclusions for the standalone application

☑ Ask Dialog ☐ Brushes Script Libraries: ☐ Database Support:

☑ Answer Dialog ☐ PDF Printer

☐ Cursors

☐ Print Dialog

Script Libraries: Browser, Browser (CEF), Database, Font Support, Geometry

Database Support: MySQL, ODBC, PostgreSQL, SqLite, Oracle

Property Profiles

● Remove all profiles on objects

○ Set all objects to profile: Other

○ Include profiles on objects and the profile library

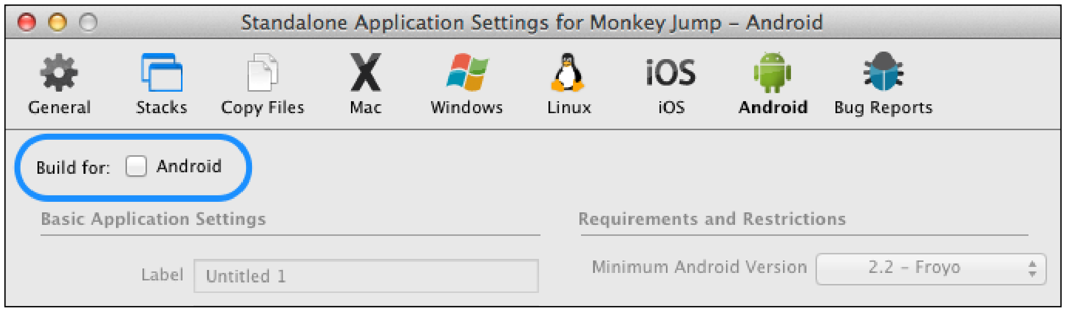
● Include all profiles ○ Only include profiles selected below

Other
Master

Default build folder: ..

☐ Automatically build here

3. Ensure that the checkbox next to the Android build option is checked. Once you select this option, a checkmark will appear in the box and the remaining options will be editable. Until you select this option, all options are disabled.



Build for: ☐ Android

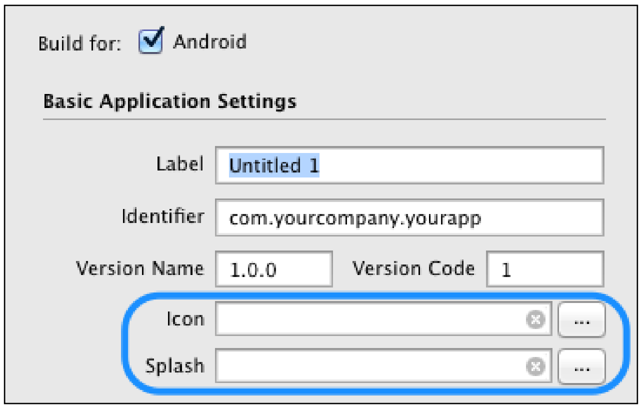
Basic Application Settings

Label: Untitled 1

Requirements and Restrictions

Minimum Android Version: 2.2 – Froyo

4. Use the **...** buttons to the right of the **Icon** and **Splash** boxes to upload your images.



Build for: ✓ Android

Basic Application Settings

Label: Untitled 1

Identifier: com.yourcompany.yourapp

Version Name: 1.0.0 Version Code: 1

Icon: ...

Splash: ...

5. Ensure that your icon image is 72 x 72 and in the PNG format.
6. You only need to upload a splash screen image if you are using a personal or educational LiveCode license. If you are using a commercial license, you do not need to upload a splash screen image. If the image is required, it should be a 600 x 600 PNG file. This splash image will be displayed on the Android screen for 5 seconds when a personal or educational license is used to develop the app.

How it works...

Pointing the file-selection window to the icon and splash screen is easy work. LiveCode does not present you with an error message if you attempt to upload an image with incorrect dimensions. So, be careful to upload the properly sized images. The files you select are embedded in your app's binary and are used when uploading it to the Google Play and Amazon Appstore.

See also

- The *Setting up your mobile environment for Android development* recipe
- The *Defining icons and images for iOS development* recipe

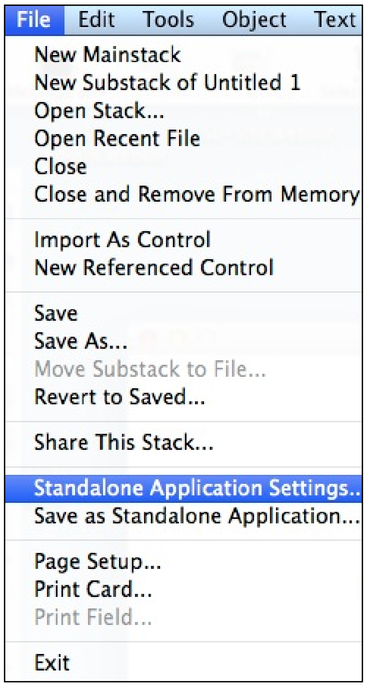
Configuring standalone application settings for iOS applications

There are several configuration settings to be considered when creating an iOS application. These settings are entered and recorded via the Standalone Application Settings dialog window. The dialog window is organized into seven sections: basic application settings, status bar, orientation options, requirements and restrictions, icons, splash screens, and custom URL scheme. This recipe addresses each setting with the exception of icons and splash screens, which are covered in other recipes in this book.

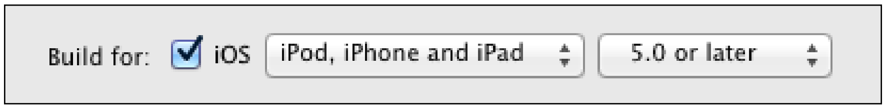
How to do it...

To configure standalone application settings for iOS applications, follow the given steps:

1. Select the **Standalone Application Settings...** option from the **File** drop-down menu.



2. In the top section of the dialog window, ensure that the checkbox labeled **iOS** (see the following screenshot) is checked.

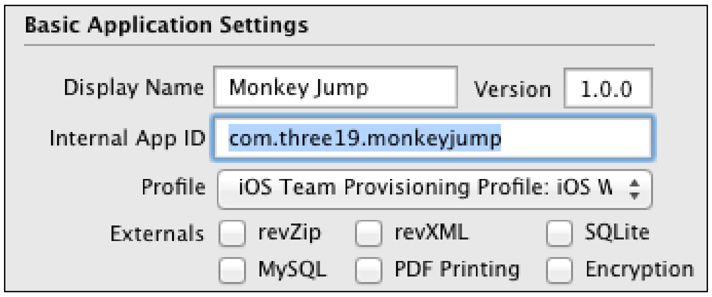


Build for: ☑ iOS | iPod, iPhone and iPad | 5.0 or later

3. Referring to the previous screenshot, select which device(s) your app will support. Your options are **iPod, iPhone and iPad**, **iPod and iPhone**, and **iPad**.
4. Referring to the previous screenshot again, select the minimum operating system that the target device(s) must have. At the time of writing this book, the options are as follows:
   - **4.3 or later**
   - **5.0 or later**
   - **5.1 or later**
   - **6.0 or later**
   - **6.1 or later**
   - **7.0 or later**
   - **7.1 or later**

Selecting earlier iOS versions will expand your potential target audience, while selecting later iOS versions will restrict a portion of your audience pool but will give you access to advanced features and functions.

5. Using the following screenshot as a reference, configure the settings on the **Basic Application Settings** page:



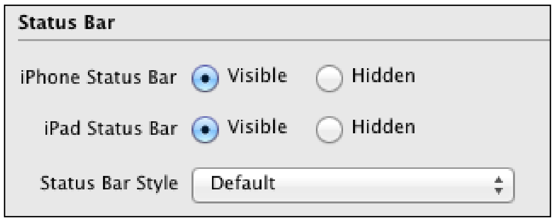
Basic Application Settings

Display Name: Monkey Jump
Version: 1.0.0
Internal App ID: com.three19.monkeyjump
Profile: iOS Team Provisioning Profile: iOS W
Externals: ☐ revZip ☐ revXML ☐ SQLite ☐ MySQL ☐ PDF Printing ☐ Encryption

The settings are explained as follows:

- **Display Name**: This is what is displayed on the SpringBoard (also referred to as the home screen) under the app's icon.
- **Version**: Your app's version number.
- **Internal App ID**: This is a universal and unique identifier that must match what you entered when you established the app's App ID in the Apple Developer portal. The format is `com.company.appname`.
- **Profile**: Select the provisioning profile you downloaded from the Apple Developer portal.
- **Externals**: Select any of the LiveCode externals (**revZip**, **revXML**, **SQLite**, **MySQL**, **PDF Printing**, and **Encryption**) based on the externals that your app requires.

6. Using the following screenshot as a reference, configure the settings on the **Status Bar** page:



Status Bar

iPhone Status Bar: [✓ Visible] Hidden

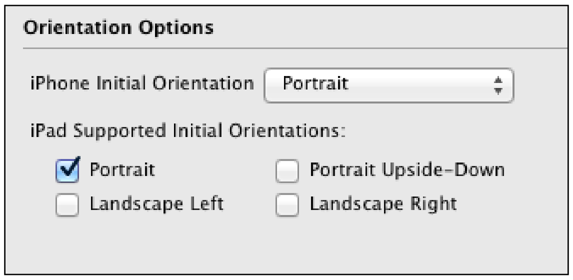
iPad Status Bar: [✓ Visible] Hidden

Status Bar Style: Default

The settings are explained as follows:

- **iPhone Status Bar**: You can have the status bar visible or hidden
- **iPad Status Bar**: You can have the status bar visible or hidden
- **Status Bar Style**: Your options here are **Default**, **Black Opaque**, and **Black Translucent**

7. Using the following screenshot as a reference, configure the settings on the **Orientation Options** page:



Orientation Options

iPhone Initial Orientation: Portrait

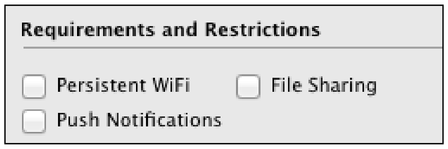
iPad Supported Initial Orientations:

- [✓] Portrait
- [ ] Portrait Upside-Down
- [ ] Landscape Left
- [ ] Landscape Right

The settings are explained as follows:

- **iPhone Initial Orientation**: You can select **Portrait**, **Portrait Upside-Down**, **Landscape Left**, or **Landscape Right** as the orientation for your app to display
- **iPad Supported Initial Orientations**: Here, you have the same selections as the previous setting, but you are able to select any/all of the orientations

8. Using the following screenshot as a reference, configure the **Requirements and Restrictions** settings:



Requirements and Restrictions

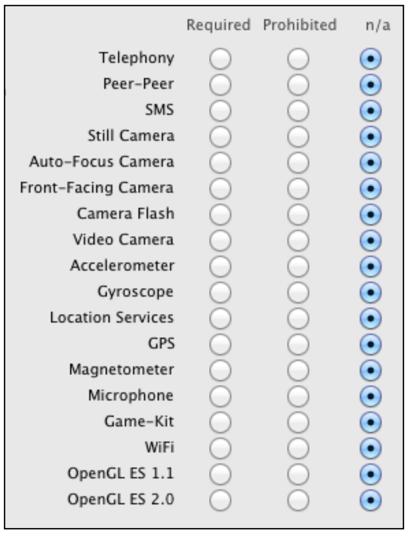
☐ Persistent WiFi ☐ File Sharing

☐ Push Notifications

The settings are explained as follows:

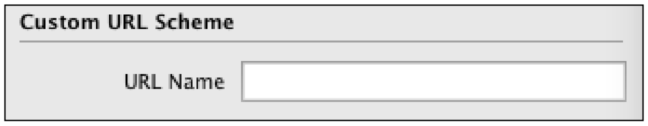
- Select whether or not your app requires persistent Wi-Fi connectivity
- Select whether or not your app requires iTunes' **File Sharing** feature to be enabled
- Select whether or not your app can receive push notifications

9. Select **Required**, **Prohibited**, or **n/a** for each of the options shown in the following screenshot. Selecting **Required** for a function or feature means that that function or feature must be present in order for the app to launch. Selecting **Prohibited** means that if that function or feature is present, the app will not get launched.



	Required	Prohibited	n/a
Telephony	○	○	●
Peer-Peer	○	○	●
SMS	○	○	●
Still Camera	○	○	●
Auto-Focus Camera	○	○	●
Front-Facing Camera	○	○	●
Camera Flash	○	○	●
Video Camera	○	○	●
Accelerometer	○	○	●
Gyroscope	○	○	●
Location Services	○	○	●
GPS	○	○	●
Magnetometer	○	○	●
Microphone	○	○	●
Game-Kit	○	○	●
WiFi	○	○	●
OpenGL ES 1.1	○	○	●
OpenGL ES 2.0	○	○	●

10. Enter, if applicable, the name of your URL scheme that was used to uniquely reference your app. Custom URL schemes can be used to enable apps to communicate with one another in order to provide services. More information on this concept can be found in Apple's iOS Developer Library.



How it works...

Using the Standalone Application Settings dialog window, we can configure a large number of settings that are relevant to our mobile application.

See also

- The *Defining icons and images for iOS development* recipe
- The *Configuring standalone application settings for Android applications* recipe

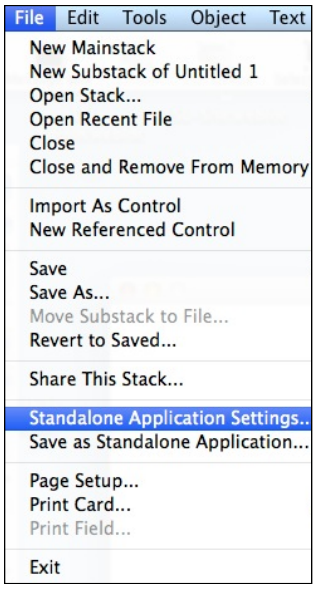
Configuring standalone application settings for Android applications

There are several configuration settings to be considered when creating an Android application. These settings are entered and recorded via the Standalone Application Settings dialog window. The dialog window is organized into five sections: basic application settings, in-app purchasing, requirements and restrictions, application permissions, and user interface options. This recipe addresses each setting.

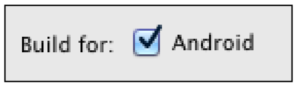
How to do it...

To configure standalone application settings for Android applications, follow the given steps:

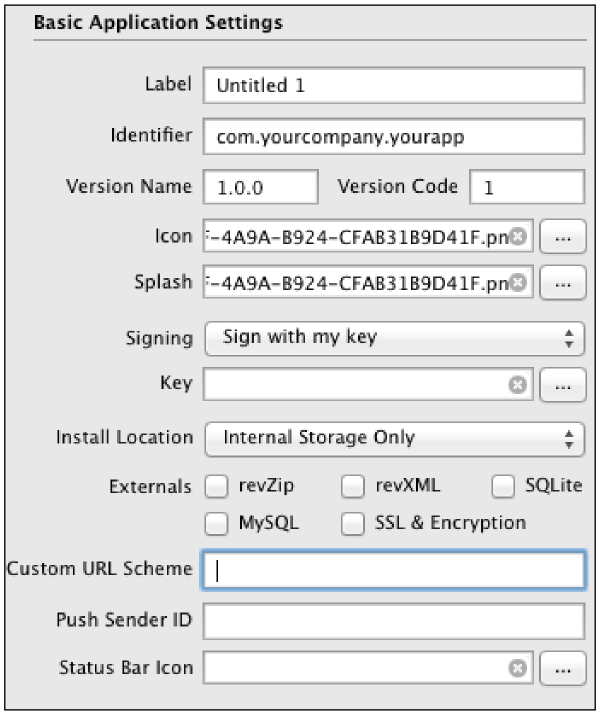
1. Select the **Standalone Application Settings...** option from the **File** drop-down menu.



2. In the top section of the dialog window, ensure that there is a check in the checkbox labeled **Android**.


Build for: ✓ Android

3. Referring to the following screenshot, determine the appropriate values for each setting:


Basic Application Settings

Label	Untitled 1
Identifier	com.yourcompany.yourapp
Version Name	1.0.0
Version Code	1
Icon	-4A9A-B924-CFAB31B9D41F.pn
Splash	-4A9A-B924-CFAB31B9D41F.pn
Signing	Sign with my key
Key	
Install Location	Internal Storage Only
Externals	revZip, revXML, SQLite, MySQL, SSL & Encryption
Custom URL Scheme	
Push Sender ID	
Status Bar Icon	

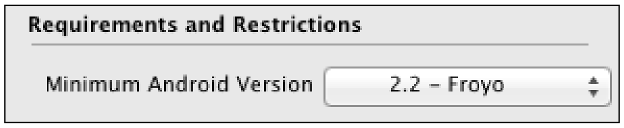
The settings are explained as follows:

- **Label**: This is what is displayed on the launcher screen under the app's icon.
- **Identifier**: This is a universal and unique identifier that uses the reverse domain name format (`com.company.appname`).
- **Version Name**: This is your app's version number.
- **Version Code**: This code is for internal version validation.
- **Icon**: This is the path to the app's icon image.
- **Splash**: This is the path to the app's splash screen image.
- **Signing**: This is where you tell LiveCode whether the APKs are to be signed with your personal key, are signed with your development key, or not at all.
- **Key**: When using the **Sign with my Key** signing option, this **Key** field points to the key-store file.
- **Install Location**: Here, you select how the app is installed on a mobile device. The options are **Internal Storage Only**, **Allow External Storage**, and **Prefer External Storage**.

- **Externals**: Select any of the LiveCode externals (**revZip**, **revXML**, **SQLite**, **MySQL**, and **SSL & Encryption**) based on the externals your app requires.
- **Custom URL Scheme**: If applicable, enter the name of your custom URL scheme that is used to uniquely reference your app.
- **Push Sender ID**: This is a unique project number associated with your app. This ID is used when you are using push notifications.
- **Status Bar Icon**: Here, you will upload your status bar icon.

4. Select whether your app will use in-app purchasing by selecting the **In App Purchasing** checkbox. If you select this option, you must identify which in-app purchasing store your app uses: Google, Samsung, or Amazon. There is additional data required if you are using Google (Public Key) or Samsung [Item Group ID and Mode (Production, Test Success, or Test Failure)].
5. Select what the minimum operating system the target device(s) must have. At the time of writing this book, the options are as follows:
   - **2.2 – Froyo**
   - **2.3 – Gingerbread**
   - **2.3.3**
   - **3.0 – Honeycomb**
   - **3.1**

   For our example, we have selected **2.2 – Froyo**.

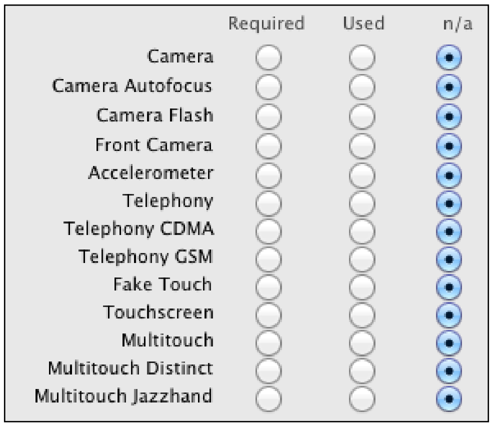


Requirements and Restrictions

Minimum Android Version: 2.2 – Froyo

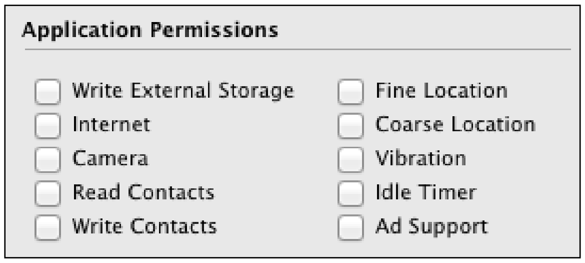
Selecting earlier Android versions will expand your potential target audience, while selecting later Android versions will restrict a portion of your audience pool but give you access to advanced features and functions.

6. Select **Required**, **Used**, or **n/a** for each of the options shown in the following screenshot. Selecting **Required** means that that function or feature will be visible to users with devices that support the feature or function. Selecting **Used** will inform the user that your app uses the function or feature.



	Required	Used	n/a
Camera	○	○	●
Camera Autofocus	○	○	●
Camera Flash	○	○	●
Front Camera	○	○	●
Accelerometer	○	○	●
Telephony	○	○	●
Telephony CDMA	○	○	●
Telephony GSM	○	○	●
Fake Touch	○	○	●
Touchscreen	○	○	●
Multitouch	○	○	●
Multitouch Distinct	○	○	●
Multitouch Jazzhand	○	○	●

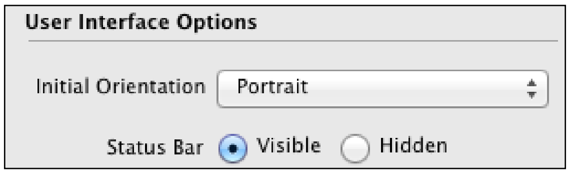
7. Select which permissions your app requires regarding the mobile device.



Application Permissions

☐ Write External Storage	☐ Fine Location
☐ Internet	☐ Coarse Location
☐ Camera	☐ Vibration
☐ Read Contacts	☐ Idle Timer
☐ Write Contacts	☐ Ad Support

8. Select **Portrait** or **Landscape** as an initial (on launch) orientation.



User Interface Options

Initial Orientation: Portrait

Status Bar: [✓ Visible] Hidden

9. Select whether you want the status bar to be visible or hidden.

How it works...

Using the Standalone Application Settings dialog window, we can configure a large number of settings relevant to our mobile application.

See also

- The *Defining icons and images for Android development* recipe
- The *Configuring standalone application settings for iOS applications* recipe

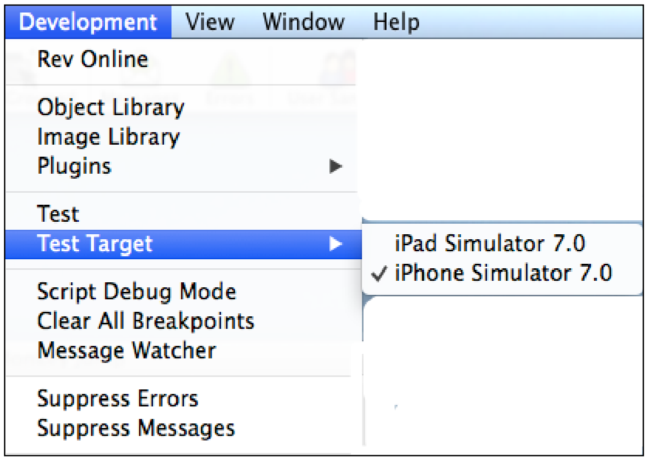
Using the simulator

Software simulators are used when we want to quickly test a mobile app or a specific function of an app in progress. Using software simulators saves a lot of time during the development process.

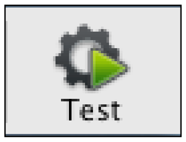
How to do it...

The following steps will take you through the usage of a simulator:

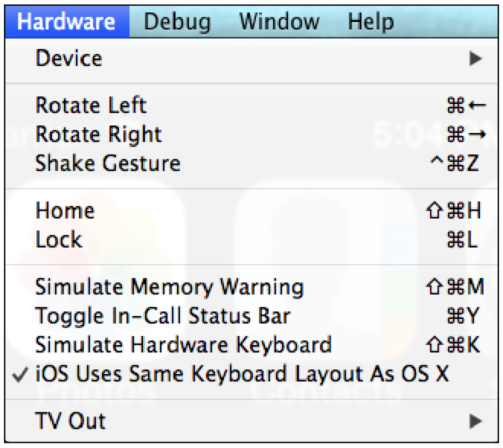
1. Select the test device from the available software simulators using the **Test Target** option of the **Development** drop-down menu.



2. Click on **Test**, which is the last icon on the LiveCode toolbar.



3. Once the simulator is running, you will have access to a **Hardware** pull-down menu, which provides you with additional options to interface with the simulator. When you select **Device**, you will be able to switch hardware devices to test.


Hardware
Debug
Window
Help
Device ▶
Rotate Left ⌘←
Rotate Right ⌘→
Shake Gesture ^⌘Z
Home ⇧⌘H
Lock ⌘L
Simulate Memory Warning ⇧⌘M
Toggle In-Call Status Bar ⌘Y
Simulate Hardware Keyboard ⇧⌘K
✓ iOS Uses Same Keyboard Layout As OS X
TV Out ▶

How it works...

Software simulators are programs that allow us to simulate actual hardware devices.

See also

- The *Setting up your mobile environment for iOS development* recipe
- The *Setting up your mobile environment for Android development* recipe

Saving a standalone mobile app

Saving a standalone mobile application involves creating the app's binary file and a supportive file-folder structure.

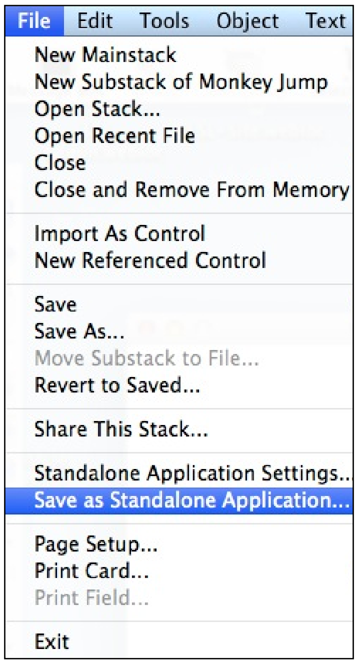
Getting ready

Before saving a standalone version of your mobile app, you will need to review and configure options presented to you via the Standalone Application Settings dialog window.

How to do it...

To save a standalone mobile app, follow the given steps:

1. Save your LiveCode project with the **Save** option under **File**, or use the keyboard shortcut *Command* + S (Mac) or *Control* + S (Windows).
2. Select the **Save as Standalone Application...** menu option.



3. Next, you will be prompted to select a location for the standalone application. The result will be a folder with subdirectories for each targeted distribution platform.

How it works...

Saving a standalone application is relatively straightforward. The important thing to remember is to review the settings in the Standalone Application Settings dialog window. When your LiveCode project is compiled, key information is taken from the selections you indicated in the aforementioned dialog window.

See also

- The *Configuring standalone application settings for iOS applications* recipe
- The *Configuring standalone application settings for Android applications* recipe

2
Human-computer Interfaces

In this chapter, we will cover the following recipes:

- Creating a new main stack
- Displaying web pages in your app
- Masking user passwords
- Including glow effects on buttons
- Including state graphics on buttons
- Getting an object's properties
- Setting custom properties
- Aligning interface objects
- Dynamically displaying interface objects
- Getting the user input
- Recording user actions
- Restricting the user input
- Using mobile keyboards
- Using a date picker
- Using a time picker
- Using effects between cards
- Using buttons for navigation

Introduction

In this chapter, you will learn how to create and control human-computer interface objects. These objects include buttons, cards, input boxes, dialog windows, and geometric shapes. You will learn how to mask passwords and how to read, create, and change an object's properties. In addition, you will learn how to allow users to navigate between your app's cards.

Creating a new main stack

One of the most fundamental tasks in LiveCode is to create a new main stack. Every LiveCode app has a main stack, at least one card, objects, and code. In this recipe, you will learn the steps required to create a new main stack.

How to do it...

To create a new main stack, follow these steps:

1. Open LiveCode.
2. From the pull-down menu, select **File**, and then select **New Main Stack**.
3. You now have a new main stack that you can start using for your mobile app. The next step is to set the stack's size properties to match the mobile environment your app will support. Using the property inspector, select **Size & Position**, and make the necessary adjustments based on the following table:

Target device	Resolution
iPhone 3GS	320 x 480
iPhone 4S	640 x 960
iPhone 5	1136 x 640
iPad (1st and 2nd Gen)	1024 x 768
iPad (3rd Gen)	2048 x 1536
iPad Mini	1024 x 768
Android (Small screen)	426 x 320
Android (Normal screen)	470 x 320
Android (Large screen)	640 x 480
Android (X-Large screen)	960 x 720

How it works...

Setting the height and width via the **Size & Position** section of the property inspector establishes the size boundaries of your mobile app. By swapping the height and width values, you can designate landscape or portrait orientation.

There's more...

Mobile device screen resolutions are subject to change with new devices, so it is important that you check what resolutions your target device supports. This information is usually available via the Apple and Android development sites.

Displaying web pages in your app

If you need to display a live web page in your mobile app, then this recipe is for you. In this recipe, you will learn how to create a display area in your app, retrieve web data, and display that data.

How to do it...

Follow these steps to display web pages in your app:

1. Create a new main stack.
2. Select the stack's card and drag an image area to display the web information.
3. Select the image and then select the **Group** icon on the toolbar to make the image a group.
4. Select the group and set the following properties:
    - **Name**: `Browser`
    - **Width**: `312`
    - **Height**: `390`
    - **Location**: `160, 225`
5. At the card level, enter the following code:
    ```
    local browserID
    ```
6. Create a `preOpenCard` handler with the following code:
    ```
      on preOpenCard
        mobileControlCreate "browser"
        put the result into browserID
        mobileControlSet browserID, "visible", \
    ```

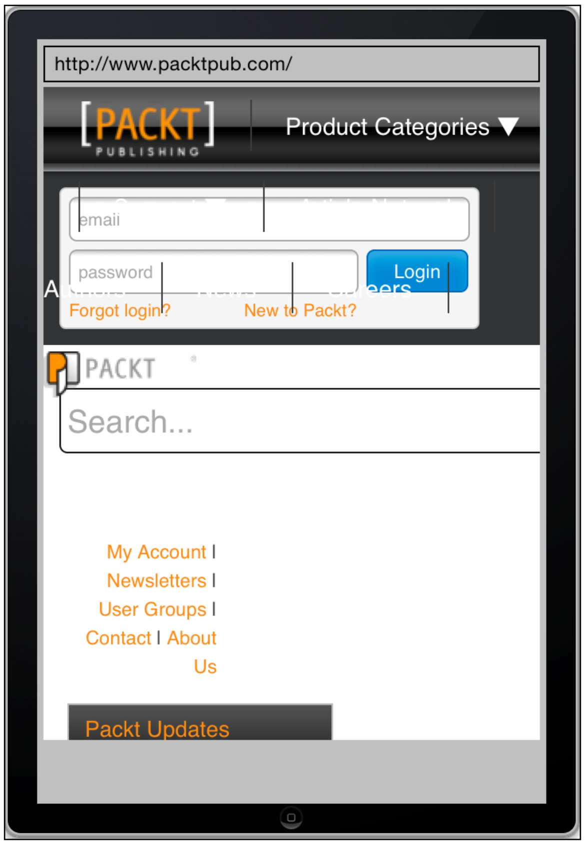
```
        "true"
      mobileControlSet browserID, "url", \
        "http://www.packtpub.com"
    end preOpenCard
```

The `preOpenCard` message is sent to a card when you first go to the card and before the `openCard` message is sent. Using the on `preOpenCard` handler allows you to manipulate a card before it is visible to the user.

7. Create a text field to display the URL. Name the field `fld_URL`.
8. Next, create the following handler:

```
on browserFinishedLoading pURL
  put pURL into field "fld_URL"
end browserFinishedLoading
```

The output is as follows:

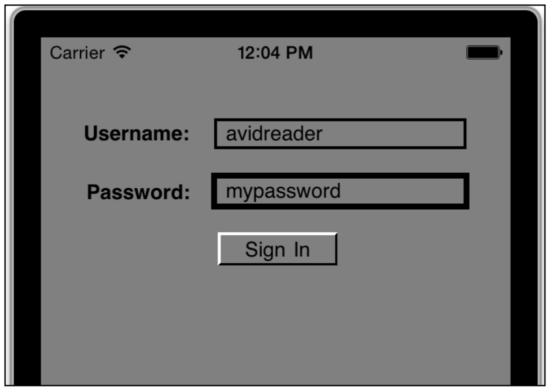


How it works...

With only two objects and a few lines of code, we are able to retrieve and display web pages from the Internet directly in our mobile apps.

Masking user passwords

When users enter their password, it should not be visible to anyone who might happen to be looking over the user's shoulder. This is an expected level of security. As shown in the following screenshot, not masking the password is unacceptable. This recipe shows you a method of masking the user's password as it is entered.



Username: avidreader

Password: mypassword

How to do it...

To accomplish our task, we will create two labels (one for the username and one for the password), three text entry fields (one for the username, a second for the unmasked password, and the third for the masked password). The masked password is the one we want displayed on the screen:

1. Create a `Username`: label for content.
2. Create a `Password`: label for content.
3. Create a text entry field and name it `fldUsername`.
4. Create a text entry field and name it `fldMaskedPassword`.
5. Create a text entry field and name it `fldUnmaskedPassword`. Set this field's visible property to `false`.

6. Add the following code to the `fldMaskedPassword` field:

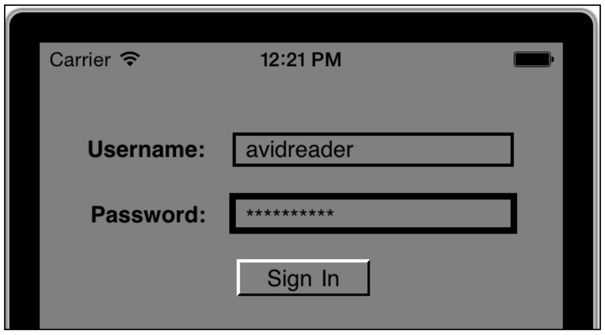
```
on keydown pKey
  put the text of fld "fldUnmaskedPassword" into tRaw
  put the text of fld "fldMaskedPassword" into tMask

  put tRaw & pKey into fld "fldUnmaskedPassword"
  put tMask & "*" into fld "fldMaskedPassword"
end keydown
```

7. When the user selects the **Log In** button, you will pull his/her username and password with statements such as the following:

```
the text of fld "fldUsername"
the text of fld "fldUnmaskedPassword"
```

How it works...

To achieve our desired results, we created an invisible text field in order to hold the user's actual password as it is entered. As each key was pressed, the key was echoed to the hidden field and displayed as ***** in the visible password field.



There's more...

If you are implementing this password-masking function in your mobile apps, you will want to thoroughly test it and ensure that you take into account user actions such as using the *Backspace*, *Delete*, and other keys. Once you have it working the way you want it, you might consider asking your friend to bug test it for you. It can be fun for people to "try and break" your app.

Including glow effects on buttons

Adding glow effects to buttons is a good way of calling attention to them and differentiating them from other buttons on your app's interface.

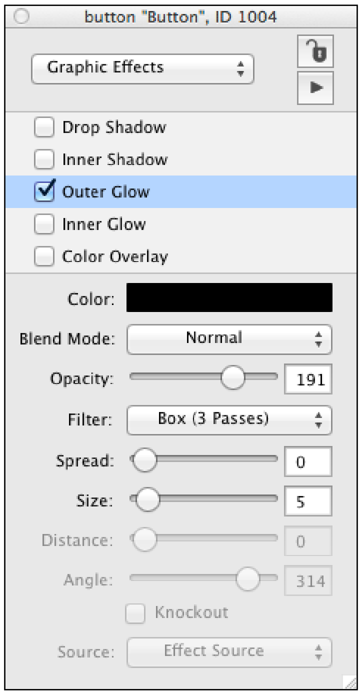
How to do it...

To include glow effects, perform the following steps:

1. Drag a button to your card.
2. Right-click on the button and select **Property Inspector**.
3. On the property inspector, select **Graphic Effects** from the pull-down menu (**Basic Properties** is the default selection).
4. Select the type of glow you desire (inner or outer).
5. Set the properties associated with the outer glow as detailed in the following table:

Property name	Property reference	Options
Color	`color`	Color palette
Blend Mode	`blendMode`	**Normal**, **Multiply**, or **Color dodge**
Opacity	`opacity`	0 – 255
Filter	`filter`	Gaussian, Box (1, 2, or 3 passes)
Spread	`spread`	0 – 255
Size	`size`	0 – 255

The following screenshot gives you an overview of these properties:



- [ ] Drop Shadow
- [ ] Inner Shadow
- [✓] Outer Glow
- [ ] Inner Glow
- [ ] Color Overlay

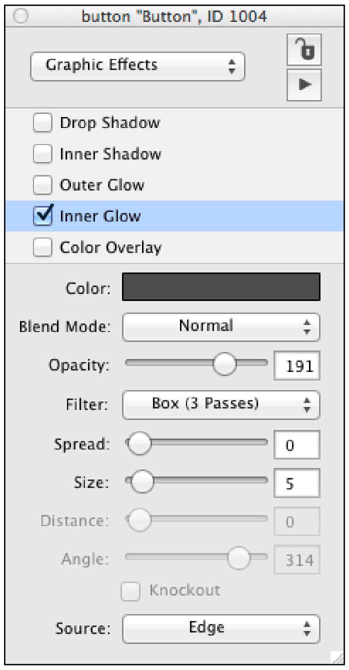
Color: ■
Blend Mode: Normal
Opacity: 191
Filter: Box (3 Passes)
Spread: 0
Size: 5
Distance: 0
Angle: 314
- [ ] Knockout

Source: Effect Source

6. Set the properties associated with the inner glow as detailed in the following table:

Property name	Property reference	Options
Color	`color`	Color palette
Blend Mode	`blendMode`	**Normal**, **Multiply**, or **Color dodge**
Opacity	`opacity`	0 – 255
Filter	`filter`	Gaussian, Box (1, 2, or 3 passes)
Spread	`spread`	0 – 255
Size	`size`	0 – 255
Source	`source`	**Edge** or **Center**

The following screenshot gives you an overview of these properties:



Color:
Blend Mode: Normal
Opacity: 191
Filter: Box (3 Passes)
Spread: 0
Size: 5
Distance: 0
Angle: 314
Source: Edge

How it works...

By using the **Graphic Effects** section of the property inspector, we can easily change the outer and inner glow properties for buttons. We can also do this programmatically by directly referencing the property reference name. For example, to change the opacity of a button, we simply use the following code:

```
set the opacity of btn "myButton" to 119
```

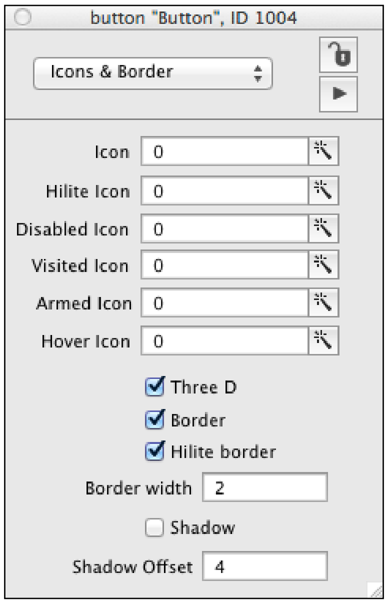
Including state graphics on buttons

Buttons have six states in LiveCode: **enabled**, **highlighted**, **disabled**, **visited**, **armed**, and **hover**. Each of these states can have an associated icon to help us communicate with our users. In this recipe, you will learn how to include state graphics on your buttons.

How to do it...

To include graphics on buttons, perform these steps:

1. Create a new main stack.
2. Drag a button to the stack's card.
3. Select, with a single click, the button you want to modify.
4. Right-click on the button and select **Property Inspector**.
5. Using the property inspector, select, **Icons & Border**. This will bring up the interface shown in the following screenshot:



button "Button", ID 1004

Icons & Border

Icon	0
Hilite Icon	0
Disabled Icon	0
Visited Icon	0
Armed Icon	0
Hover Icon	0

[✓] Three D
[✓] Border
[✓] Hilite border

Border width: 2

[ ] Shadow

Shadow Offset: 4

6. Select the button state you want to modify. An explanation of each state is in the following table:

Button State	Graphic reference	This state exists when...
Enabled	Icon	The button is available for use.
Highlighted	Hilite icon	The button is being depressed.
Disabled	Disabled icon	The button is not available for use.
Visited	Visited icon	The button has already been used by the user.
Armed	Armed icon	The mouse pointer moves into the button.
Hover	Hover icon	The mouse pointer is over the button.

7. If you know the object ID number of the graphic you want to use, enter it directly. Otherwise, you can use the browse tool to the right of the text entry field.

How it works...

We can tell LiveCode which images are to be used for each button state by referring to the graphic's object ID. As the button's state changes, so will its graphic (icon).

Getting an object's properties

LiveCode's development environment comes with several basic types of objects: button, checkbox, tab panel, label, field, data grid, menu, progress and scrollbar, slider, image, and graphic. For some of these basic types, there are several subtypes (for example, for menus, there is: dropdown, option, pop-up, and combo box). It is often necessary to obtain specific properties for evaluation in our apps. This recipe shows us how this is done.

How to do it...

Perform the following steps to get an object's properties:

1. Use the `get` command in LiveCode to obtain an object's property. For example, use the following code to get a button's label:

   ```
   get the label of btn "myButton"
   ```

2. To use a property, you can put it into a temporary variable for later use:

   ```
   local tText
   put the label of btn "myButton" into tText
   ```

3. You can also get an object's property as part of a conditional statement such as in the following example that evaluates a button's label:

   ```
   if the label of btn "myButton" is "Sally" then
     // do something
   else
     // do something else
       end if
   ```

How it works...

All of LiveCode's objects have multiple properties. In order to determine what properties exist, we need to refer to the property inspector. When we hover over a property in that interface, we are provided with the property's name via a tooltip. When using the property inspector, remember that there are several sections (**Basic**, **Icons & Border**, **Colors & Patterns**, **Geometry**, **Graphic effects**, **Blending**, **Property profiles**, **Size & Position**, and **Text formatting**).

See also

- The *Including glow effects on buttons* recipe
- The *Setting custom properties* recipe

Setting custom properties

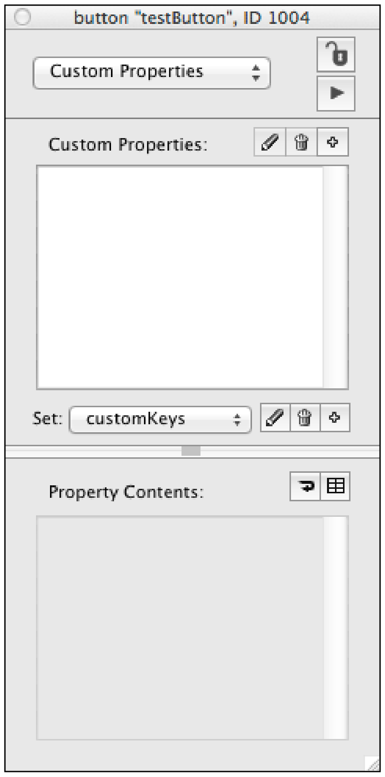
LiveCode's objects come with an impressive number of properties that help us control how they look and function. We can also add our own properties to objects called custom properties. This provides us with a tremendous amount of flexibility.

How to do it...

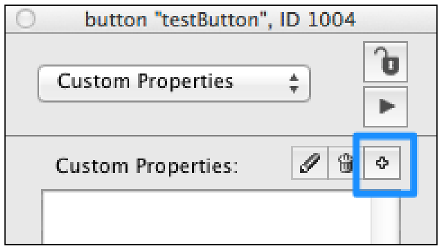
Follow these steps to set the custom properties:

1. Create a main stack.
2. Drag any object onto the stack's card (for example, a button).
3. Select an object that you wish to add custom properties to.
4. Right-click on the object and select **Property Inspector**.

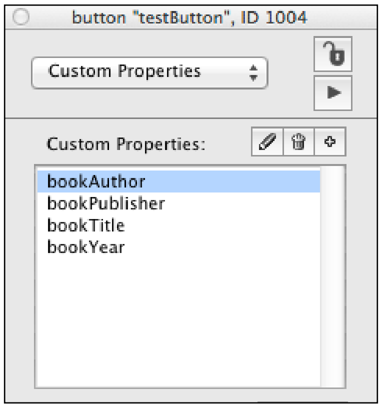
5. Using the property inspector, select **Custom Properties**. You should see an interface similar to the one displayed in the following screenshot:



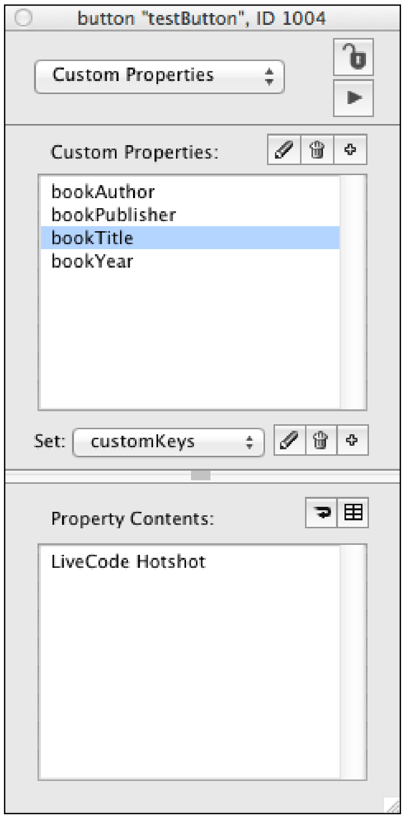
6. Click on the plus sign to add a new custom property:



7. Enter a name for the custom property and click on **OK**. For example, you might want to use a button to represent a book in a game. Add the custom properties, `bookTitle`, `bookYear`, `bookAuthor`, `bookPublisher`, until your display matches the following screenshot:



8. Next, we will add content to the custom properties. There are two ways to do this: direct entry and programmatically. We can use the property inspector interface to directly enter the content by selecting the custom property and then typing the content in the **Property Contents** area (see the following screenshot).



LiveCode Hotshot

9. Alternatively, you can add the content to the custom properties via code:

```
set the bookPublisher of btn "testButton" to "Packt"
```

How it works...

We used the property inspector to create custom properties and add content. This content can be changed during an app's execution via code as we demonstrated in step 9.

See also

- The *Getting an object's properties* recipe

Aligning interface objects

The mobile apps we develop can have several objects on the screen simultaneously. It is important that these objects line up and not look haphazard. LiveCode provides us with tools to make aligning our objects relatively easy.

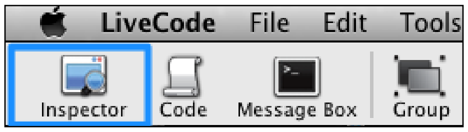
How to do it...

Perform the following steps to align interface objects:

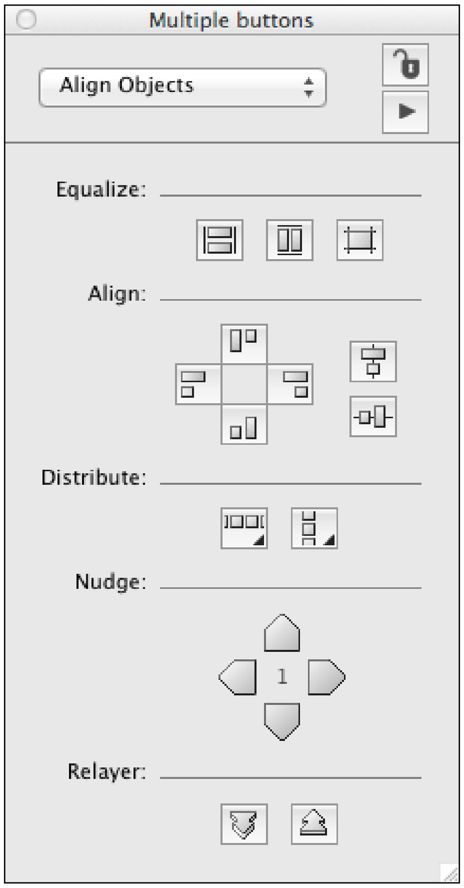
1. Create a new main stack.
2. Drag two or more objects on the stack's card. For example, drag two button objects to the card.
3. Select the objects you want to align. To do this, click on the first object, and then, while holding the *Shift* key, click on the remaining objects you want to align. When you are finished selecting the objects, release the *Shift* key.

> When selecting multiple objects to align, it is important that you ensure the first object you select is the one you want to use as a reference. For example, you might want to align selected objects by the left ledge of the first object you selected.

4. Click on the Inspector icon on the toolbar. This will bring up the property inspector.



5. Select the **Align Objects** section of the property inspector:



6. Select one or more of the six alignment buttons (top, bottom, left, right, middle, or center) to align your objects.

How it works...

Aligning the objects via the property inspector saves us time when creating and laying out the human-computer interface.

Dynamically displaying interface objects

Oftentimes, we will create interface objects and only have them appear on the screen when appropriate. For example, you might have a graphic indicator that the user has unread system messages. If the value is 1 or higher, then you might have the graphic visible; otherwise, you might hide it. This recipe shows you how to accomplish this task.

How to do it...

Perform the following steps to dynamically display interface objects:

1. Create a new main stack.
2. Drag a button to the stack's card.
3. Change the name of the new button to `testButton`.
4. To make an object, such as a button, visible, use the following syntax:

   ```
   set the visible of <object type> <"object name"> to true
   ```

   So, for example, if you have a button named `testButton`, your code will be as follows:

   ```
   set the visible of btn "testButton" to true
   ```

5. To hide the example button from step 4, enter the following code:

   ```
   set the visible of btn "testButton" to false
   ```

6. To toggle a button's visibility, add the following code to the button:

   ```
   if the visible of me is true then
     set the visible of me to false
   else
     set the visible of me to true
   end if
   ```

How it works...

We can use an object's `visible` property to dynamically display or hide the object by setting `visible` to `true` or `false`.

Getting the user input

We want users to interact with our mobile apps. This means that we must provide the opportunity for the user to provide input, we must capture that input, and we must process it. This recipe focuses on capturing the user input via the mobile keyboard.

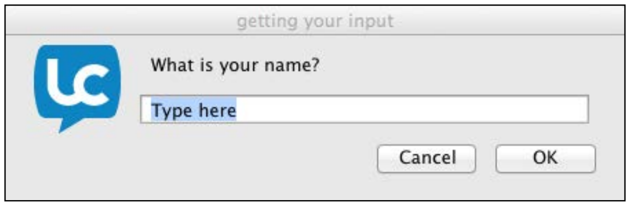
How to do it...

The following steps will help you get the user input:

1. Create a new main stack.
2. Drag a new button to the stack's card.
3. Assign the following code to the new button:

```
on mouseUp
  ask "What is your name?" with "Type here" \
    titled "getting your input"
end mouseUp
```

4. Execute the code to see the results.



How it works...

We used the `ask` command to provide the prompt (**What is your name**) and default text (**Type here**). The results are put into the `it` variable. This provides us the opportunity to evaluate and manipulate the user's input. If the user selects the **Cancel** button, no value is put into the `it` variable. You can test this by using the `if it is empty` line of code.

Recording user actions

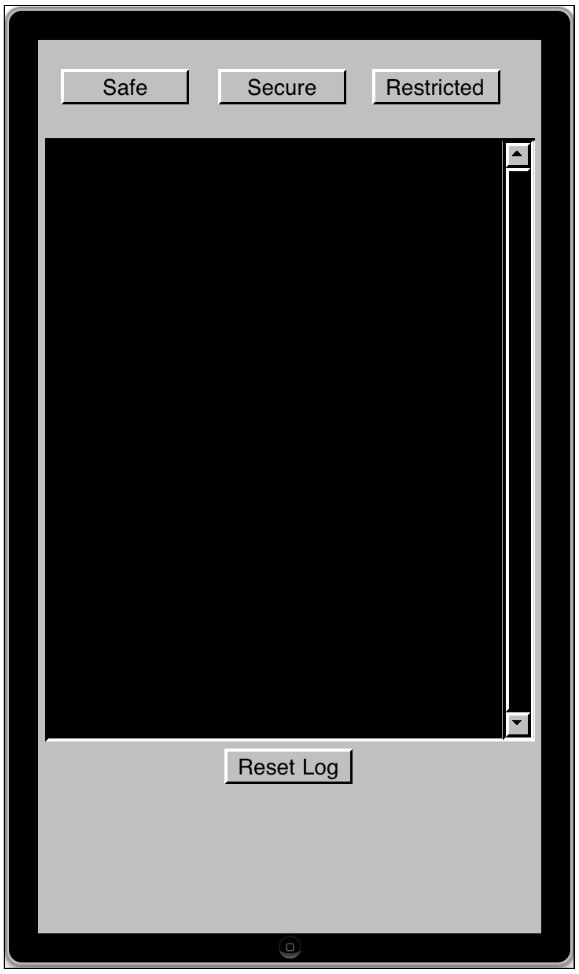
Some applications are better served when you record the user's actions. In this context, a user action refers to an in-app behavior such as clicking on/touching a button or moving a slider. In this recipe, we will create a user action log and program a button to record user actions.

How to do it...

To record user actions, perform the following steps:

1. Create a new main stack for a mobile application.
2. Add three buttons across the top named `Safe`, `Secure`, and `Restricted`.
3. Add a button named `Reset Log` at the bottom-center of the card.

4. Create a scrolling text field named `fldLog` to fill the remainder of the card.
5. Set the background color of the `fldLog` field to black.
6. Set the foreground color of the `fldLog` field to yellow. This will be the color of the text entered into the log.
7. Set text size of the `fldLog` field to **14**.
8. Set the `traversalOn` property to `false` (deselect the **Focusable** checkbox in the **Basic Properties** section of the property inspector).
9. When you complete steps 1 to 5, your interface should look similar to the following screenshot. Make any adjustments to your four buttons and the scrolling text field before moving on to step 7.



10. Add the following code to the card that contains your interface:

```
global logLine

on preCardOpen
  put 1 into logLine
end preCardOpen

on updateLog msg2log
  put msg2log into line logLine of fld "fldLog"
  add 1 to logLine
end updateLog
```

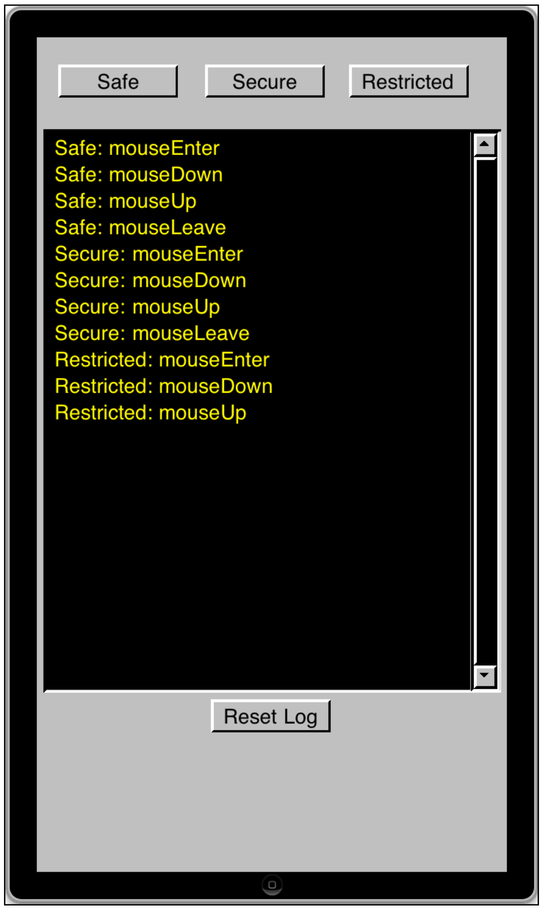
11. Add the following code to the **Reset Log** button:

```
on mouseUp
  global logLine

  put 1 into logLine
  put empty into fld "fldLog"
end mouseUp
```

12. Add the following code to the **Safe**, **Secure**, and **Restricted** buttons:

```
on mouseUp
  updateLog(short name of me & ": mouseUp")
end mouseUp

on mouseDown
  updateLog(short name of me & ": mouseDown")
end mouseDown

on mouseEnter
  updateLog(short name of me & ": mouseEnter")
end mouseEnter

on mouseLeave
  updateLog(short name of me & ": mouseLeave")
end mouseLeave
```

13. Test your application. It should look similar to the following screenshot:



```
Safe: mouseEnter
Safe: mouseDown
Safe: mouseUp
Safe: mouseLeave
Secure: mouseEnter
Secure: mouseDown
Secure: mouseUp
Secure: mouseLeave
Restricted: mouseEnter
Restricted: mouseDown
Restricted: mouseUp
```

How it works...

By creating listeners (also known as handlers), we can record user actions in our apps. Depending upon your application, you might want to create a system log for security purposes or even to provide a display (as we did in our example) for users to review.

Restricting the user input

There is an old saying: garbage in and garbage out. This refers to users entering invalid data into a computerized system. Fortunately, we have some control over what users can input. This recipe shows you how this is done.

How to do it...

To restrict user input, perform the following steps:

1. Create a new main stack.
2. Drag a text input box to the stack's card.
3. Add the following code to the text input box:

```
on keyDown theKey
  if theKey is in "abcdefghijklmnopqrstuvwxyz" then
    pass keyDown
  else
     beep
    end if
  end keyDown
```

How it works...

By evaluating the user's input prior to allowing it to pass to the next level in the message chain, we can selectively accept or reject it.

There's more...

In this recipe, we intercepted messages from the keyboard using our on `keyDown` handler. In LiveCode, messages are triggered by events such as the `mouseDown` message being sent when the user selects/clicks on an object. In this example, the event was the clicking of the object and `mouseDown` was the message. Messages have an informational path in that they are first heard or are available at the object, group (if applicable), card, stack, and finally, at the LiveCode Engine level.

Using mobile keyboards

In order to enable a mobile keyboard, we often just need the user to touch an input area. For example, if a user taps on a username input field, the keyboard will automatically appear. This recipe covers options regarding mobile keyboard activation.

How to do it...

Follow the given steps to use mobile keyboards:

1. Create a new main stack.
2. Drag a text input field to the stack's card.

3. Enter one of the following lines of code, at the card level, depending upon which type of keyboard you want displayed:

   ```
   mobileSetKeyboardType "alphabet"
   mobileSetKeyboardType "numeric"
   ```

4. On the text input field, ensure that the **Focusable** checkbox is checked (set `traversalOn` to `true`).
5. Test the code in the simulator or on an actual device.

How it works...

The iOS and Android operating systems support autokeyboard activation. LiveCode takes advantage of this. So, all you need to do is to identify which type of keyboard you want displayed.

Using a date picker

It is easier for users to input dates via a date picker where they are presented with scroll wheels. This has become a standard user interface object and is expected by users. This recipe shows you how to use a date picker in LiveCode.

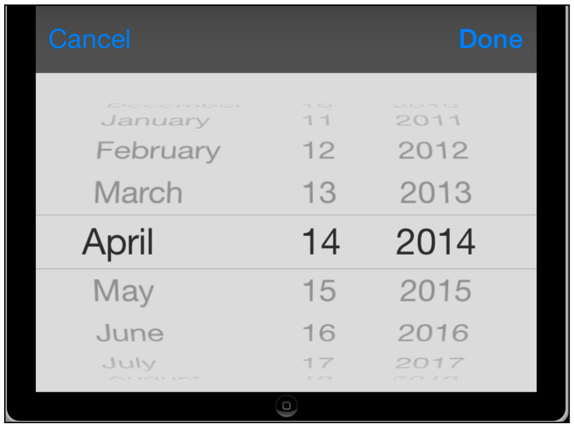
How to do it...

To use a date picker, follow the given steps:

1. Create a new main stack.
2. Drag a button to the stack's card and assign the following properties:
   - **Name**: `Launch Date Picker`
   - **Width**: `184`
   - **Height**: `23`
   - **Location**: `200, 33`
3. Add the following code to the new button to initiate a date picker, as displayed in the given screenshot:

   ```
   on mouseUp
     mobilePickDate "date"
   end mouseUp
   ```

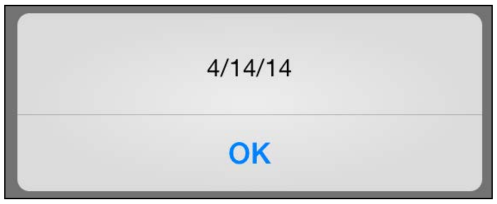
The output will be as follows:



4. When the user selects the date (month, day, year) and taps **Done**, you will want to ensure that you have a way to capture the date. LiveCode puts the user's selection into the `the result` system variable. You can test this by adding one line of code to the `mobilePickDate` command:

```
on mouseUp
  mobilePickDate "date"
  answer the result
end mouseUp
```

The output will be as follows:



How it works...

We used the `mobilePickDate` command to instantiate the date picker on a mobile device. We are provided with the user's selection in the `the result` system variable. This allows us to analyze and manipulate the input.

See also

- The *Using a time picker* recipe

Using a time picker

It is easier for users to input the time via a time picker where they are presented with scroll wheels. This has become a standard user interface object and is expected by users. This recipe shows you how to use a time picker in LiveCode.

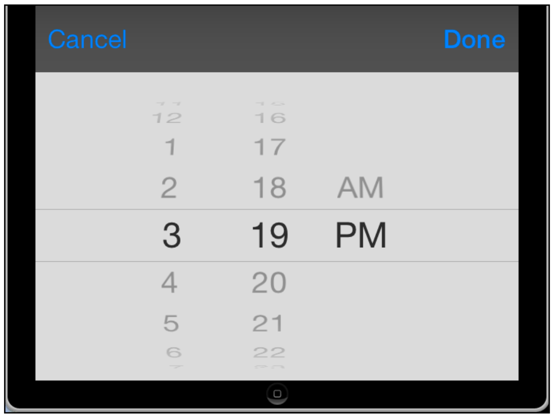
How to do it...

To use a time picker, follow the given steps:

1. Add the following code to initiate a date picker, as displayed in the given screenshot:

```
on mouseUp
  mobilePickDate "time"
end mouseUp
```

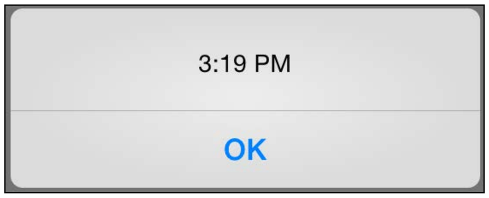
   The output will be as follows:



2. When the user selects the time (hours, minutes, a.m./p.m.) and taps **Done**, you will want to ensure that you have a way to capture the time. LiveCode puts the user's selection into the `the result` system variable. You can test this by adding a line of code to the `mobilePickDate` command:

```
on mouseUp
  mobilePickDate "time"
  answer the result
end mouseUp
```

The output will be as follows:


3:19 PM

How it works...

We used the `mobilePickDate` command to instantiate the time picker on a mobile device. We are provided with the user's selection in the `the result` system variable. This allows us to analyze and manipulate the input.

See also

- The *Using a date picker* recipe

Using effects between cards

LiveCode allows us to use visual effects between cards. There are several transition effects that can provide your app with a desirable visual effect.

How to do it...

To use effects between cards, follow the given steps:

1. Create a new main stack.
2. Change the name of the stack's card to `Blue`.
3. Add a second card and name it `Red`.
4. Change the background color of the **Blue** card to blue.
5. Change the background color of the **Red** card to red.
6. Drag a new button to the **Blue** card.
7. Add the following code to the new button on the **Blue** card:

```
on mouseUp
  visual effect dissolve
  go to card "Red"
end mouseUp
```

8. Add a button to the **Red** card.
9. Add the following code to the new button on the **Red** card:

```
on mouseUp
  visual effect push up
  go to card "Blue"
end mouseUp
```

10. Run the app in the simulator to see the results.

How it works...

To implement visual transitions between cards, we use the following syntax:

```
visual effect <effect>
go to card <card>
```

When we call the `visual effect` command, we are telling LiveCode what visual transition we want on the next card transition. There are many visual effects available, including the following:

- Push up
- Push down
- Push right
- Push left
- Reveal up
- Reveal down
- Reveal left
- Reveal right
- Scroll up
- Scroll down
- Scroll left
- Scroll right
- Curl up
- Curl down
- Flip left
- Flip right

We can also add speed (very slow, slow, normal, fast, or very fast) to our visual effect command. For example, the following code:

```
visual effect dissolve very fast
```

There's more...

Normally, a mobile app will only use one or two transition effects. Using too many transition effects might result in a less-than-polished-looking app.

Using buttons for navigation

Buttons are frequently used to navigate between cards. Standard buttons include ones labeled **Back** and **Next**. This recipe shows you how to program these buttons.

How to do it...

To navigate using buttons, follow the given steps:

1. Create a new main stack.
2. Drag a new button to the stack's card.
3. Change the name of the new button to `Next` and add the following code to it:
   ```
   go to next card
   ```
4. Create a button named `Back` and add the following code to it:
   ```
   go back 1
   ```

How it works...

Card navigation in LiveCode is relatively simply. All of the following commands will work:

```
go to card "Main"
go to card 3
go to next card
go back 1
```

3
Loops and Timers

In this chapter, we will cover the following topics:

- Implementing a countdown timer
- Implementing a count-up timer
- Pausing a timer
- Resuming a timer
- Using a loop to count
- Using a loop to iterate through a list

Introduction

In this chapter, you will learn how to use timers and loops in your mobile apps. Timers can be used for many different functions, including a basketball shot clock, car racing time, the length of time logged into a system, and so much more. Loops are useful for counting and iterating through lists. All of this will be covered in this chapter's recipes.

Implementing a countdown timer

To implement a countdown timer, we will create two objects: a field to display the current timer and a button to start the countdown. We will code two handlers: one for the button and one for the timer.

How to do it...

Perform the following steps to create a countdown timer:

1. Create a new main stack.
2. Place a field on the stack's card and name it `timerDisplay`.
3. Place a button on the stack's card and name it `Count Down`.
4. Add the following code to the `Count Down` button:

```
on mouseUp
  local pTime

  put 19 into pTime
  put pTime into fld "timerDisplay"
  countDownTimer pTime
end mouseUp
```

5. Add the following code to the `Count Down` button:

```
on countDownTimer currentTimerValue
  subtract 1 from currentTimerValue
  put currentTimerValue into fld "timerDisplay"
  if currentTimerValue > 0 then
    send "countDownTimer" && currentTimerValue to me
      in 1 sec
  end if
end countDownTimer
```

6. Test the code using a mobile simulator or an actual device.

How it works...

To implement our timer, we created a simple callback situation where the `countDownTimer` method will be called each second until the timer is zero. We avoided the temptation to use a `repeat` loop because that would have blocked all other messages and introduced unwanted app behavior.

There's more...

LiveCode provides us with the `send` command, which allows us to transfer messages to handlers and objects immediately or at a specific time, such as this recipe's example.

See also

- The *Implementing a count-up timer* recipe
- The *Pausing a timer* recipe
- The *Resuming a timer* recipe

Implementing a count-up timer

To implement a count-up timer, we will create two objects: a field to display the current timer and a button to start the upwards counting. We will code two handlers: one for the button and one for the timer.

How to do it...

Perform the following steps to implement a count-up timer:

1. Create a new main stack.
2. Place a field on the stack's card and name it `timerDisplay`.
3. Place a button on the stack's card and name it `Count Up`.
4. Add the following code to the `Count Up` button:

```
on mouseUp
  local pTime

  put 0 into pTime
  put pTime into fld "timerDisplay"
  countUpTimer pTime
end mouseUp
```

5. Add the following code to the `Count Up` button:

```
on countUpTimer currentTimerValue
  add 1 to currentTimerValue
  put currentTimerValue into fld "timerDisplay"
  if currentTimerValue < 10 then
    send "countUpTimer" && currentTimerValue to me
      in 1 sec
  end if
end countUpTimer
```

6. Test the code using a mobile simulator or an actual device.

How it works...

To implement our timer, we created a simple callback situation where the `countUpTimer` method will be called each second until the timer is at 10. We avoided the temptation to use a `repeat` loop because that would have blocked all other messages and introduced unwanted app behavior.

There's more...

Timers can be tricky, especially on mobile devices. For example, using the `repeat` loop control when working with timers is not recommended because `repeat` blocks other messages.

See also

- The *Implementing a countdown timer* recipe
- The *Pausing a timer* recipe
- The *Resuming a timer* recipe

Pausing a timer

It can be important to have the ability to stop or pause a timer once it is started. The difference between stopping and pausing a timer is in keeping track of where the timer was when it was interrupted. In this recipe, you'll learn how to pause a timer. Of course, if you never resume the timer, then the act of pausing it has the same effect as stopping it.

How to do it...

Use the following steps to create a count-up timer and pause function:

1. Create a new main stack.
2. Place a field on the stack's card and name it `timerDisplay`.
3. Place a button on the stack's card and name it `Count Up`.
4. Add the following code to the `Count Up` button:

```
on mouseUp
  local pTime
  put 0 into pTime
  put pTime into fld "timerDisplay"
  countUpTimer pTime
end mouseUp
```

5. Add the following code to the `Count Up` button:

```
on countUpTimer currentTimerValue
  add 1 to currentTimerValue
  put currentTimerValue into fld "timerDisplay"
  if currentTimerValue < 60 then
    send "countUpTimer" && currentTimerValue to me
    in 1 sec
  end if
end countUpTimer
```

6. Add a button to the card and name it `Pause`.
7. Add the following code to the `Pause` button:

```
on mouseUp
  repeat for each line i in the pendingMessages
    cancel (item 1 of i)
  end repeat
end mouseUp
```

In LiveCode, the `pendingMessages` option returns a list of currently scheduled messages. These are messages that have been scheduled for delivery but are yet to be delivered.

8. To test this, first click on the **Count Up** button, and then click on the **Pause** button before the timer reaches 60.

How it works...

We first created a timer that counts up from 0 to 60. Next, we created a **Pause** button that, when clicked, cancels all pending system messages, including the call to the `countUpTimer` handler.

See also

- The *Implementing a countdown timer* recipe
- The *Implementing a count-up timer* recipe
- The *Resuming a timer* recipe

Resuming a timer

If you have a timer as part of your mobile app, you will most likely want the user to be able to pause and resume a timer, either directly or through in-app actions. See previous recipes in this chapter to create and pause a timer. This recipe covers how to resume a timer once it is paused.

How to do it...

Perform the following steps to resume a timer once it is paused:

1. Create a new main stack.
2. Place a field on the stack's card and name it `timerDisplay`.
3. Place a button on the stack's card and name it `Count Up`.
4. Add the following code to the `Count Up` button:

```
on mouseUp
  local pTime

  put 0 into pTime
  put pTime into fld "timerDisplay"
  countUpTimer pTime
end mouseUp
on countUpTimer currentTimerValue
  add 1 to currentTimerValue
  put currentTimerValue into fld "timerDisplay"
  if currentTimerValue < 60 then
    send "countUpTimer" && currentTimerValue to me
    in 1 sec
  end if
end countUpTimer
```

5. Add a button to the card and name it `Pause`.
6. Add the following code to the `Pause` button:

```
on mouseUp
  repeat for each line i in the pendingMessages
    cancel (item 1 of i)
  end repeat
end mouseUp
```

7. Place a button on the card and name it `Resume`.

8. Add the following code to the `Resume` button:

```
on mouseUp
  local pTime

  put the text of fld "timerDisplay" into pTime
  countUpTimer pTime
end mouseUp

on countUpTimer currentTimerValue
  add 1 to currentTimerValue
  put currentTimerValue into fld "timerDisplay"
  if currentTimerValue <60 then
    send "countUpTimer" && currentTimerValue to me
    in 1 sec
  end if
end countUpTimer
```

9. To test this, first, click on the **Count Up** button, then click on the **Pause** button before the timer reaches 60. Finally, click on the **Resume** button.

How it works...

We first created a timer that counts up from 0 to 60. Next, we created a **Pause** button that, when clicked, cancels all pending system messages, including the call to the `countUpTimer` handler. When the **Resume** button is clicked on, the current value of the timer, based on the `timerDisplay` button, is used to continue incrementing the timer.

In LiveCode, `pendingMessages` returns a list of currently scheduled messages. These are messages that have been scheduled for delivery but are yet to be delivered.

See also

- The *Implementing a countdown timer* recipe
- The *Implementing a count-up timer* recipe
- The *Pausing a timer* recipe

Using a loop to count

There are numerous reasons why you might want to implement a counter in a mobile app. You might want to count the number of items on a screen (that is, cold pieces in a game), the number of players using your app simultaneously, and so on. One of the easiest methods of counting is to use a loop. This recipe shows you how to easily implement a loop.

How to do it...

Use the following steps to instantiate a loop that counts:

1. Create a new main stack.
2. Rename the stack's default card to `MainScreen`.
3. Drag a label field to the card and name it `counterDisplay`.
4. Drag five checkboxes to the card and place them anywhere. Change the names to `1`, `2`, `3`, `4`, and `5`.
5. Drag a button to the card and name it `Loop to Count`.
6. Add the following code to the `Loop to Count` button:

```
on mouseUp
  local tButtonNumber

  put the number of buttons on this \
  card into tButtonNumber
  if tButtonNumber > 0 then
    repeat with tLoop = 1 to tButtonNumber
      set the label of btn value(tLoop) to \
      "Changed " & tLoop
    end repeat
    put "Number of button's changed: " & \
    tButtonNumber into fld "counterDisplay"
  end if
end mouseUp
```

7. Test the code by running it in a mobile simulator or on an actual device.

How it works...

In this recipe, we created several buttons on a card. Next, we created code to count the number of buttons and a repeat control structure to sequence through the buttons and change their labels.

See also

- The *Using a loop to iterate through a list* recipe

Using a loop to iterate through a list

In this recipe, we will create a loop to iterate through a list of text items. Our list will be a to-do or action list. Our loop will process each line and number them on screen. This type of loop can be useful when you need to process lists of unknown lengths.

How to do it...

Perform the following steps to create an iterative loop:

1. Create a new main stack.
2. Drag a scrolling list field to the stack's card and name it `myList`.
3. Change the contents of the `myList` field to the following, paying special attention to the upper- and lowercase values of each line:
   - `Wash Truck`
   - `Write Paper`
   - `Clean Garage`
   - `Eat Dinner`
   - `Study for Exam`
4. Drag a button to the card and name it `iterate`.
5. Add the following code to the `iterate` button:

```
on mouseUp
  local tLines

  put the number of lines of fld "myList" into tLines
  repeat with tLoop = 1 to tLines
    put tLoop & " - " & line tLoop of fld
    "myList"into line tLoop of fld "myList"
  end repeat
end mouseUp
```

6. Test the code by clicking on the **iterate** button.

How it works...

We used the `repeat` control structure to iterate through a `list` field one line at a time. This was accomplished by first determining the number of lines in that `list` field, and then setting the `repeat` control structure to sequence through the lines.

See also

- The *Using a loop to count* recipe

4
Managing Text

In this chapter, we will cover the following recipes:

- Reading the user input
- Searching text
- Replacing text
- Combining text
- Encrypting text
- Writing text
- Reading text
- Sorting text
- Formatting text
- Appending text
- Translating text into Pig Latin

Introduction

A large number of mobile applications require us to deal with text in various formats.
For example, we might need to read and evaluate user input to ensure that they entered a valid e-mail address, or we might use user text for searching other text. LiveCode provides a tremendous ability for us to work with text. In this chapter, you will learn how to accomplish tasks related to working with text. You will learn how to read user input and how to replace, combine, and encrypt text. You will also learn how to write, read, sort, format, and append text. Lastly, you will learn how to translate text into Pig Latin.

Reading the user input

In this recipe, you will learn how to prompt the user for input and how to read that input. This can be useful in many situations. To accomplish this task, we will ask the user for his/her name, read his/her input, and output a greeting to let the user know that we were paying attention.

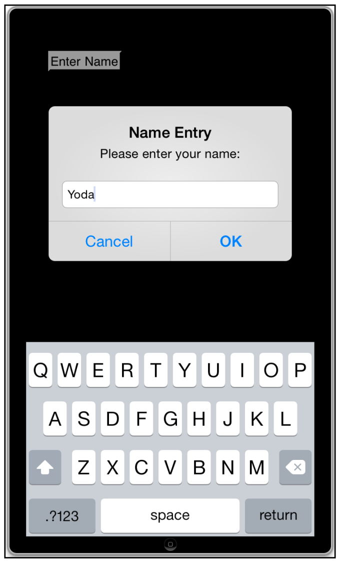
How to do it...

Reading the user input is a fundamental function of most mobile apps. Follow the given steps to learn how to prompt the user for text and then read it:

1. Create a new main stack in LiveCode.
2. Drag a button to the main card.
3. Name the new button `getName`.
4. Change the label of the button to `Enter Name`.
5. Set the background color (fill color) of the new button to white.
6. Add the following code to the button:

```
on mouseUp
  ask question "Please enter your name:" titled
    "Name Entry"
end mouseUp
```

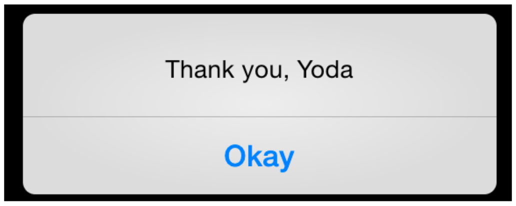
7. When you run the application in the mobile simulator, you should see the interface shown in the following screenshot:



8. Now, let's modify the button's code to read the input and display it in a message to the user. Change the `getName` button's code to match the following:

```
on mouseUp
  local userName

  ask question "Please enter your name:" titled
    "Name Entry"
  put it into userName
  answer "Thank you, " & userName with "Okay"
end mouseUp
```

9. When you run the application in the simulator, you should see a pop-up dialog that includes the user's name:


Thank you, Yoda

How it works...

We used both the `ask` and `answer` commands in our recipe. First, the `ask` command prompted the user for input. The resultant input was placed in the `it` variable. We copied the input into our own variable (`userName`) and used it with the `answer` command.

See also

- The *Restricting the user input* recipe in *Chapter 2*, *Human-computer Interfaces*

Searching text

In this recipe, we will prompt the user for some text and then prompt him/her for text to search for. We will evaluate the first set of text to determine if the second bit of text is contained within the first. We will provide the user with the results.

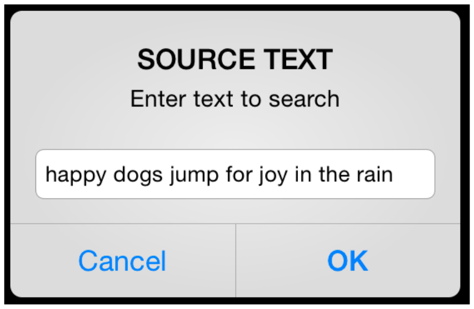
How to do it...

Follow this recipe's steps to create functionality that asks for input of the text that is to be searched for and the text that is to be analyzed with the search criteria:

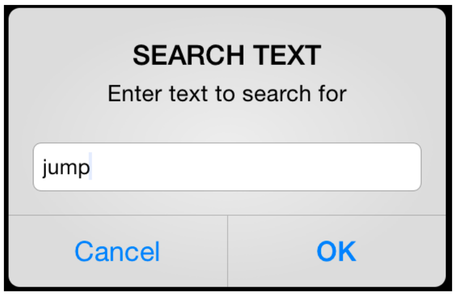
1. Create a new main stack in LiveCode.
2. Drag a button to the card.
3. Add the following code to the new button:

```
on mouseUp
  local sourceText, searchText

  ask question "Enter text to search:" titled "SOURCE TEXT"
  put it into sourceText
  --
  ask question "Enter text to search for:" titled
    "SEARCH TEXT"
  put it into searchText
  --
  if searchText is among the words of sourceText then
  answer "text found"
  else
    answer "text not found"
  end if
end mouseUp
```

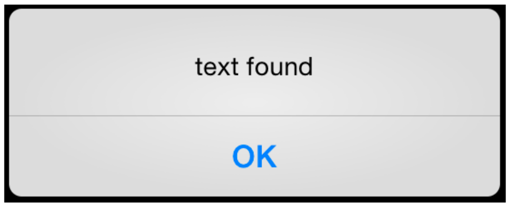
4. Run the application in a simulator and enter the text shown in the following screenshot:


SOURCE TEXT
Enter text to search
happy dogs jump for joy in the rain

5. After clicking on **OK**, enter the text shown in the following screenshot:



6. After clicking on **OK** on the search text dialog, you should see the results as illustrated in the following screenshot:



How it works...

Searching for text within text can be relatively straightforward using LiveCode. In our recipe, we used the `is among the words` code to search for text in another string of text.

There's more...

You might also wish to designate the `caseSensitive` property to `true` or `false`. This property determines if string comparisons are case-sensitive or case-insensitive. The default setting is `false`. To set the property to `true`, use:

```
set the caseSensitive to true
```

To set the property to `false`, use:

```
set the caseSensitive to false
```

Replacing text

In this recipe, you will learn how to replace portions of text based on a search and match schema. We will use the `replace` command to accomplish this task.

How to do it...

Replacing sections of text can be a powerful part of a mobile app. Follow this recipe's steps to perform a global find-and-replace operation:

1. Create a new main stack in LiveCode.
2. Drag a new button to the card.
3. Name the new button `replaceText`.
4. Add the following code to the button:

```
on mouseUp
  local tempText
  put "EMP001, EMP002, EMP003, EMP004, EMP005" into
    tempText
  replace "EMP" with "EMP#:" in tempText
  answer tempText
end mouseUp
```

How it works...

We created a local variable (`tempText`) to hold our original text. Next, we used the `replace` command to change all occurrences of `EMP` to `EMP#:`. We then used the `answer` command to display the results.

Combining text

In this recipe, you will learn how to take two text values and combine them into one.

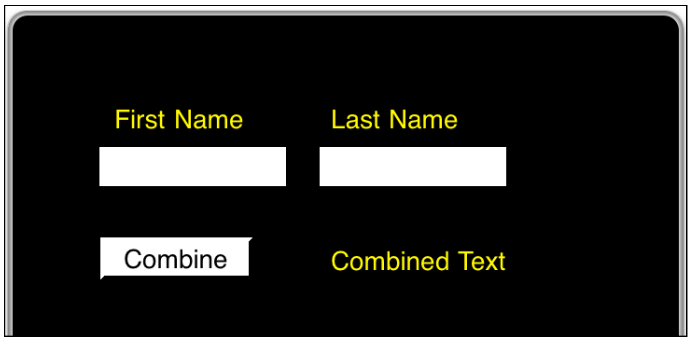
How to do it...

Follow this recipe's steps to take two user-entered bits of text and combine them into one:

1. Create a new main stack in LiveCode.
2. Set the background color of the stack's card to black.
3. Add a first label with the following properties to the card:
    - **Name**: `fldFirst`

- foregroundColor: Yellow
- **Contents**: `First Name`

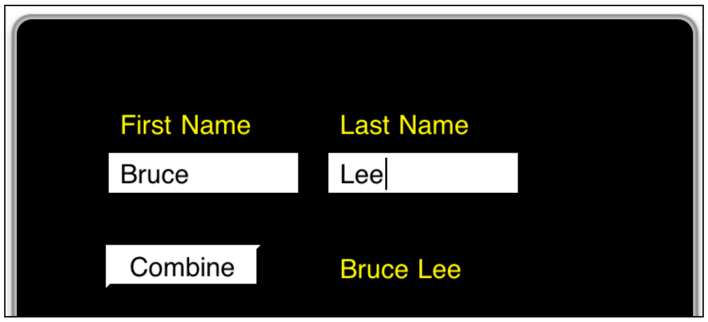
4. Add a second label with the following properties to the card:
   - **Name**: `fldLast`
   - foregroundColor: Yellow
   - **Contents**: `Last Name`
5. Add a third label with the following properties to the card:
   - **Name**: `fldCombined`
   - **Width**: `200`
   - foregroundColor: Yellow
   - **Contents**: `Combined Text`
6. Add a first text input field to the card with the following properties:
   - **Name**: `firstName`
   - foregroundColor: Black
   - backgroundColor: White
   - borderWidth: `0`
7. Add a second text input field to the card with the following properties:
   - **Name**: `lastName`
   - foregroundColor: Black
   - backgroundColor: White
   - borderWidth: `0`
8. Create a button named `Combine`.
9. Layout the three labels, the two text input boxes, and one button as illustrated in the following screenshot:



10. Add the following code to the **Combine** button:

```
on mouseUp
  local fName, lName, newCombinedName

  put the text of fld "firstName" into fName
  put the text of fld "lastName" into lName
  put fName && lName into newCombinedName
  put newCombinedName into fld "fldCombined"
end mouseUp
```

11. Run the application in the mobile simulator and enter the values `Bruce` and `Lee` for the first and last name respectively. You should see the same results as shown in the following screenshot:



How it works...

We used the `&&` operator to combine two strings and automatically add a space in between them. If we simply used the `&` operator, then the first and last name would have been run together, without a space. Of course we could have alternatively used the following:

```
put fName & " " & lName into newCombinedName
```

Encrypting text

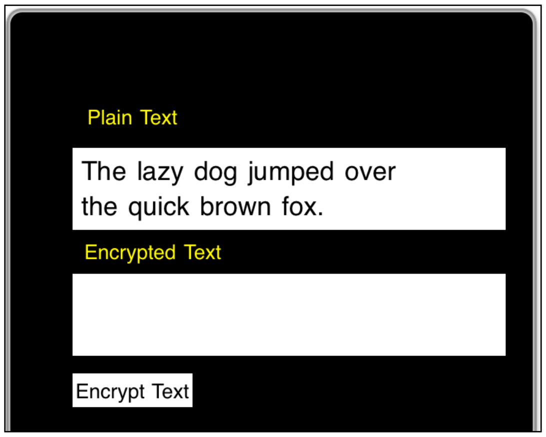
LiveCode includes several cyphers that allow us to encrypt and decrypt data. In this recipe, you will select a cipher, assign a password, and encrypt data. You will also decrypt that data using the same cypher and password.

How to do it...

Follow this recipe's steps to encrypt and decrypt text:

1. Create a new main stack in LiveCode.
2. Set the background color of the stack's card to black.
3. Add a label with the following properties:
   - **Name**: `fldPlain`
   - **Width**: `175`
   - foregroundColor: Yellow
   - **contents**: `Plain Text`
4. Add a second label with the following properties:
   - **Name**: `fldEncrypted`
   - **Width**: `175`
   - foregroundColor: Yellow
   - **Contents**: `Encrypted Text`
5. Add a scrolling text field with the following properties:
   - **Name**: `plainText`
   - traversalOn (**Focusable**): Keep it unchecked
   - showFocusBorder (**Focus border**): Keep it unchecked
   - borderWidth: `0`
   - foregroundColor: Black
   - backgroundColor: White
   - **Contents**: `The lazy dog jumped over the quick brown fox.`
   - Size of text: `18`
6. Add a second scrolling text field with the following properties:
   - **Name**: `encryptedText`
   - traversalOn (**Focusable**): Keep it unchecked
   - showFocusBorder (**Focus border**): Keep it unchecked
   - borderWidth: `0`
   - foregroundColor: Black
   - backgroundColor: White
   - Size of text: `18`

7. Add a button with the following properties:
   - **Name**: `btnEncrypt`
   - **Label**: `Encrypt Text`
   - foregroundColor: Black
   - backgroundColor: White
   - borderWidth: `0`
8. Layout the labels, the text fields, and the button as shown in the following screenshot:

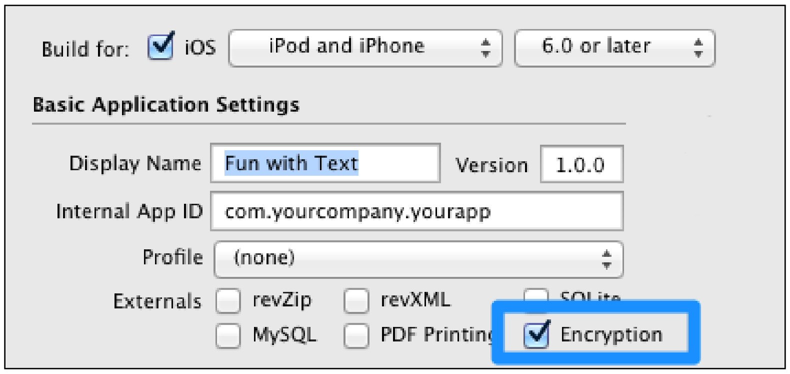


Plain Text

The lazy dog jumped over the quick brown fox.

Encrypted Text

9. Access the Standalone Application Settings dialog from the **File** drop-down menu. Ensure that there is a check in the **Encryption** checkbox, as shown in the following screenshot:



Build for: ✓ iOS iPod and iPhone 6.0 or later

Basic Application Settings

Display Name: Fun with Text

Version: 1.0.0

Internal App ID: com.yourcompany.yourapp

Profile: (none)

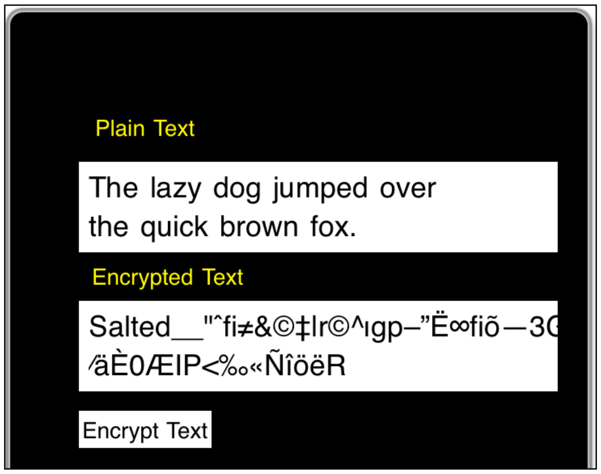
Externals: ☐ revZip ☐ revXML ☐ SQLite ☐ MySQL ☐ PDF Printing ✓ Encryption

[When working with Android apps, select the **SSL & Encryption** checkbox in the Standalone Application Settings dialog.]

10. Add the following code to the `BtnEncrypt` button:

```
on mouseUp
  local tCypherList, sourceText

  put the cipherNames into tCypherList
  put the text of fld "plaintext" into sourceText
  encrypt sourceText using rc4 with password coolbeans
  put it into fld "encryptedText"
end mouseUp
```

11. Run the application in the mobile simulator and click on the button labeled **Encrypt Text**. The results should be similar to what is shown in the following screenshot:



The lazy dog jumped over the quick brown fox.

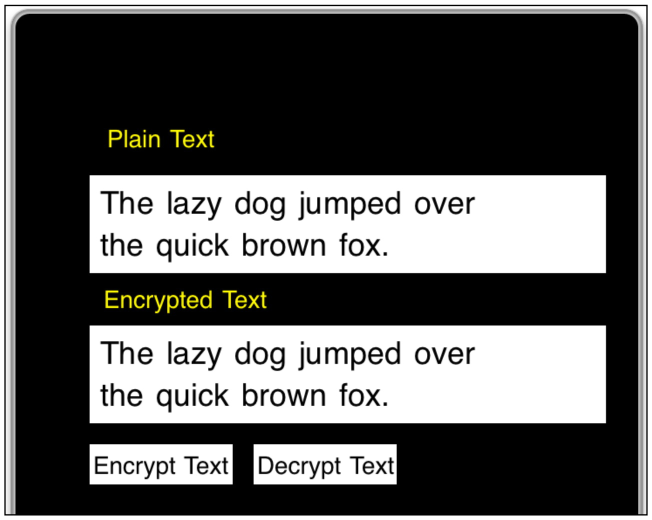
Salted__"ˆfi≠&©‡Ir©^ıgp–"Ë∞fiõ—3G
⁄âÈ0ÆIP<‰«ÑîöëR

12. Add a second button to the right of the current `btnEncrypt` button with the following properties:
    - **Name**: `btnDecrypt`
    - **Label**: `Decrypt Text`
    - foregroundColor: Black
    - backgroundColor: White
    - borderWidth: `0`

13. Add the following code to the `btnDecrypt` button:

```
on mouseUp
  local tCypherList, encryptedText

  put the cipherNames into tCypherList
  put the text of fld "encryptedText" into encryptedText
  decrypt encryptedText using rc4 with password coolbeans
  put it into fld "encryptedText"
end mouseUp
```

14. Run the application in the mobile simulator and click on the button labeled **Decrypt Text**. The results should be similar to what is shown in the following screenshot:



The lazy dog jumped over the quick brown fox.

The lazy dog jumped over the quick brown fox.

How it works...

In this recipe, we encrypted and decrypted text using a cipher and password. The cipher came from the list of cyphers available via the system variable `cipherNames`. In step 9, we marked the encryption external to ensure that external is bundled with the standalone versions of your mobile app.

Writing text

When coding mobile applications, you might have the need to write text to the screen or external files, such as displaying a level completion message or posting a new high score on a leaderboard. In this recipe, you will learn how to write text to the screen of a mobile device.

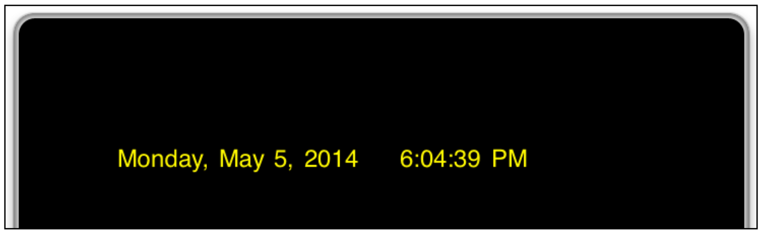
How to do it...

Follow the steps in this recipe to write text on the user's screen:

1. Create a new main stack in LiveCode.
2. Add a label field to the card with the following properties:
    - **Name**: `userInput`
    - **Width**: `300`
    - foregroundColor: Yellow
    - Align text left
    - Size of text: `18`
3. Add the following code to the card:

```
on openCard
  put the long date & tab & the long time into fld
    "userInput"
end openCard
```

4. Run the app in the mobile simulator. Your results should be similar to what is displayed in the following screenshot:



Monday, May 5, 2014	6:04:39 PM

How it works...

In this recipe, we used the `put` command to put data on the screen using a text field. The same command can be used to put data in objects such as buttons, text fields, and so on.

See also

- The *Reading text* recipe

Reading text

In this recipe, you will learn how to read text that is present on the mobile device such as data entered by the user.

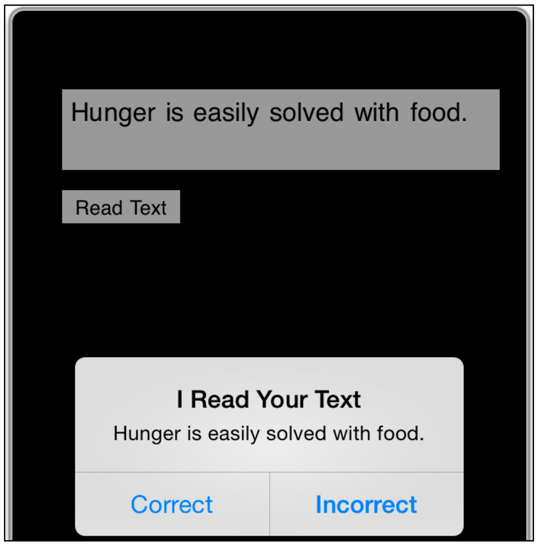
How to do it...

Follow this recipe's steps to read text from the mobile device's screen:

1. Create a new main stack in LiveCode.
2. Add a label field to the card with the following properties:
   - **Name**: `userInput`
   - **Width**: `300`
   - foregroundColor: Black
   - Align text left
3. Add a button to the card with the following properties:
   - **Name**: `btnRead`
   - **Label**: `Read Text`
   - foregroundColor: Black
   - backgroundColor: White
   - borderWidth: `0`
4. Add the following code to the `btnRead` button:

```
on mouseUp
  answer the text of fld "userInput" with "Correct" \
and "Incorrect" titled "I Read Your Text"
end mouseUp
```

5. Run the app in the mobile simulator. Enter `Hunger is easily solved with food.` in the input text field. Your results should match what is displayed in the following screenshot:


Hunger is easily solved with food.

I Read Your Text
Hunger is easily solved with food.

How it works...

In this recipe, we read the text entered by the user as part of the `answer` command using the statement `the text of fld`.

See also

- The *Writing text* recipe

Sorting text

LiveCode provides us with the ability to easily sort data in both ascending and descending order. In this recipe, we will sort a list of US states in ascending order.

How to do it...

Follow the steps in the recipe to sort text:

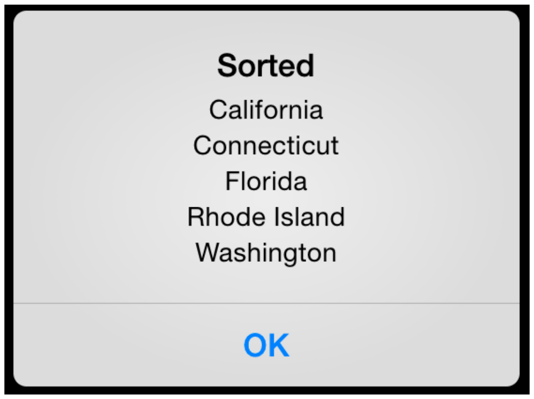
1. Create a new main stack in LiveCode.
2. Add a button to the card with the following properties:
    - **Name**: `btnSort`
    - **Label**: `Sort`

- foregroundColor: Black
- backgroundColor: White
- borderWidth: `0`

3. Add the following code to the `btnSort` button:

```
on mouseUp
  local someStates
  put "Connecticut" into line 1 of someStates
  put "California" into line 2 of someStates
  put "Washington" into line 3 of someStates
  put "Florida" into line 4 of someStates
  put "Rhode Island" into line 5 of someStates
  sort lines of someStates ascending text
  answer someStates titled "Sorted"
end mouseUp
```

4. Run the app in the mobile simulator. As illustrated in the following screenshot, the states have been properly sorted in ascending order:


Sorted

California
Connecticut
Florida
Rhode Island
Washington

How it works...

We used the `sort` command to sort the lines of our data. We provided three parameters to the command: container, direction, and type of sort. The container was our `someStates` variable, which consisted of five lines of text. The direction was ascending and the type of sort was text. It is simply a matter of changing `ascending` to `descending` in the script if you want to sort in the descending order.

Formatting text

In this recipe, you will learn how to format text displayed on a screen. By following this recipe's steps, you will change the entire text, sections, and subsections of text.

How to do it...

Follow the given steps to format the onscreen text:

1. Create a new main stack in LiveCode.
2. Drag a scrolling text field to the card.
3. Name the new field `myText`.
4. Enter three or more paragraphs of text in the contents of the field.

   [You can autogenerate dummy text at `http://www.lipsum.com/feed/html`.]

5. Now that the text is in place, let's change the background color to yellow. In the message box (accessible via the **Tools** drop-down menu), type in the following code and press *Enter*:

```
set the backgroundColor of fld "myText" to yellow
```

6. Next, we will change the background color of the first paragraph to red. Enter the following code in the message box:

```
set the backgroundColor of line 1 of fld "myText" to red
```

7. Let's add a left and right indentation to the second paragraph by entering the following code in the message box:

```
set the leftIndent of line 2 of fld "myText" to 25
set the rightIndent of line 2 of fld "myText" to 25
```

8. Next, we will add a black border of size 3 to the third paragraph. Execute the following code in the message box:

```
set the bordercolor of line 3 of fld "myText" to black
set the borderwidth of line 3 of fld "myText" to 3
```

9. Lastly, we will change the background color of words 5 through 10 of the second paragraph to violet. We will do this with the following code in the message box:

```
set the backgroundColor of word 5 to 10 of line 2 of fld
  "myText" to violet
```

How it works...

Text can be viewed in a similar way to objects in LiveCode, in that they have properties that can be changed. In this recipe, we changed the following properties of the text: backgroundColor, borderColor, borderWidth, leftIndent, and rightIndent.

Appending text

In this recipe, we will learn how to add text to the end of the existing text. In our context, appending text refers to adding text after the existing text, or at the end of the text.

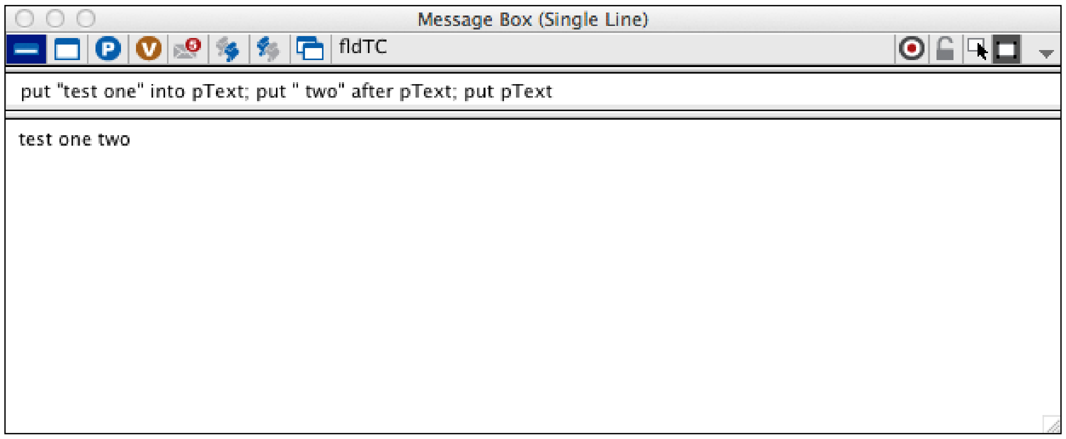
How to do it...

Appending text in LiveCode is a very simple task. Follow the given two steps to append one string (text) with another.

1. Open the message box and enter the following code:

```
put "test one" into pText; put " two" after pText; put
  pText
```

2. When you execute the code in the message box, you should see the following result:



```
put "test one" into pText; put " two" after pText; put pText
```

```
test one two
```

How it works...

To accomplish our task of appending text, we used the `after` keyword. LiveCode made the task easy by allowing us to, simply put, "place this after that".

Translating text into Pig Latin

In this recipe, you will learn how to translate standard text into Pig Latin. To accomplish this task, you will ask the user for an English word, determine if the word starts with a vowel or not, and then convert the word to Pig Latin in two steps.

> Pig Latin words are formed by modifying English words. The first consonant of an English word is moved to the end of the word and "ay" is added as a suffix. If the word begins with a vowel, then it is modified only by adding the suffix "ay." So, "LiveCode" would become "iveCodelay".

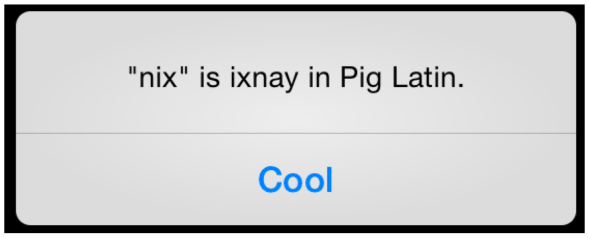
How to do it...

While you might never need to use Pig Latin in the mobile apps you develop, this recipe shows some of LiveCode's powerful text controls:

1. Create a new main stack in LiveCode.
2. Create a button with the name `pigLatin`.
3. Add the following code to the **pigLatin** button:

```
on mouseUp
  local englishWord, pigLatinWord, tLetter

  ask question "Enter an English word" titled "Word Entry"
  put it into englishWord
  put englishWord into pigLatinWord
  # check if word starts with a vowel
  if char 1 of pigLatinWord is not among the chars \
  of "aeiou" then
  // word does not start with a vowel
    put char 1 of pigLatinWord into tLetter
    delete char 1 of pigLatinWord
    put tLetter after pigLatinWord
  end if
  # Add "ay" to end of word
  put "ay" after pigLatinWord

  answer quote & englishWord & quote & " is " &quote &
    pigLatinWord & quote \
  & " in Pig Latin." with "Cool"
end mouseUp
```

4. Test your code in the mobile simulator and enter `nix` for your English word. Your results should be the same as shown in the following screenshot:


"nix" is ixnay in Pig Latin.

How it works...

To translate English words to Pig Latin, we first evaluated the user input to determine whether the first letter of his/her word began with a vowel or a consonant. If it started with a consonant, then we recorded the first character, deleted it from the word, and added it to the end. Next, we added the "ay" suffix. Lastly, we provided output to the user, showing him/her the original word and the new Pig Latin word.

There's more...

In this recipe, we created code that handles English words that start with a vowel or a single consonant. For complete and accurate translations, all leading consonants should be moved to the end of the word prior to adding the "ay" suffix. So, the word "style" would become "estylay".

5 Communications

In this chapter, we will cover the following topics:

- Initiating a phone call
- Sending an e-mail
- Formatting an e-mail

Introduction

In this chapter, you will learn how to initiate a phone call, send an e-mail, and format an e-mail using LiveCode. These tasks have become common features of most mobile apps. The recipes in this chapter will arm you with all you need to integrate this functionality into your own mobile apps.

Initiating a phone call

You might be developing a mobile application that requires users to be able to make a phone call to a specific number. For example, you might have a hotline for animal abuse. In this recipe, you will learn how to use LiveCode to instantiate the mobile device's phone software and dial a number.

How to do it...

Perform the following steps to prepare your mobile app to initiate a phone call:

1. Create a new LiveCode main stack.
2. Set the background color of the default card to black.

3. Drag a new button to the card and set the following properties:
    - **Name**: `btn_callHotline`
    - **Label**: `Call Hotline`
    - **Width**: `144`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - hiliteBorder: Keep it unchecked
    - backgroundColor: White
4. Add the following code to the `btn_callHotline` button:

```
on mouseUp
  launch url "tel:+19998887777"
end mouseUp
```

5. Test this application on an actual device.

How it works...

We just need to make a call to the `launch url` command and pass it the telephone number and `tel:` prefix.

There's more...

Most simulators will not simulate the use of a telephone, so you will need to compile this app and save it as a standalone application. You can then test it on an actual device.

Sending an e-mail

Mobile devices often support e-mail clients. We can use LiveCode to interact with the mobile device's e-mail system. This recipe illustrates the relative ease in sending an e-mail from within a mobile app developed with LiveCode.

How to do it...

Follow the given steps to send an e-mail:

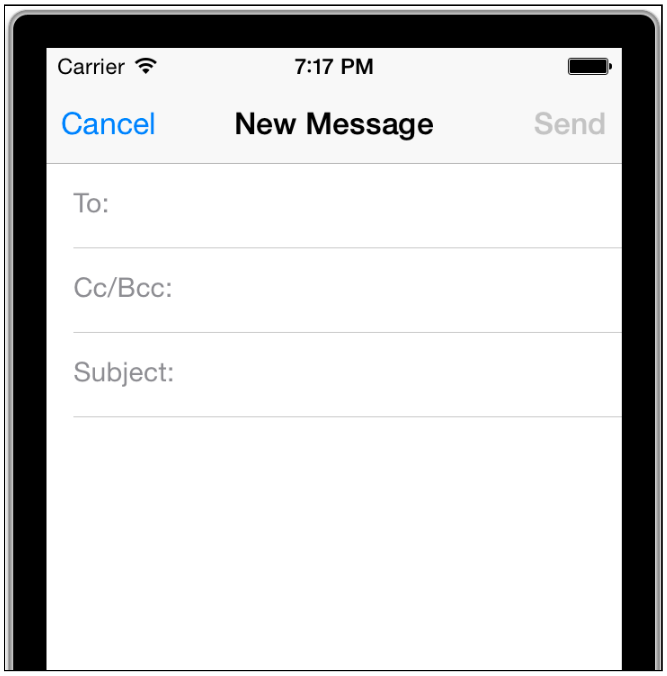
1. Create a new LiveCode stack.
2. Set the background color of the default card to black.
3. Drag a new button to the card and set the following properties:
    - **Name**: `btn_sendEmail`
    - **Label**: `Send Email`

- **Width**: `144`
- threeD: Keep it unchecked
- showBorder: Keep it unchecked
- hiliteBorder: Keep it unchecked
- backgroundColor: White

4. Add the following code to the `btn_sendEmail` button:

```
on mouseUp
  revMail
end mouseUp
```

5. When you run the app in the simulator and click on the **Submit** button, you should see the e-mail client appear, as illustrated in the following screenshot:



How it works...

LiveCode makes our work easy when it comes to invoking the e-mail client. We made a call to the `revMail` command and the e-mail client's **New Message** dialog was instantiated.

See also

- The *Formatting an e-mail* recipe

Formatting an e-mail

It can be helpful to users if you prepopulate e-mails before presenting the user with the mobile operating system's e-mail dialog interface. You might, for example, want to create an e-mail message highlighting an in-game achievement of your users. This recipe will show you how to format an e-mail using the To address, CC address, e-mail subject, and e-mail body.

How to do it...

By performing the following steps, you can have your mobile app prepopulate some e-mail properties so that it is preformatted or prefilled for your user:

1. Create a new LiveCode stack.
2. Set the background color of the default card to black.
3. Drag a new label to the card and set the following properties:
    - **Name**: `lbl_toAddress`
    - foregroundColor: White
    - textSize: **14**
    - Text style: Bold
    - **Contents**: `To Address`
4. Drag a new label to the card and set the following properties:
    - **Name**: `lbl_ccAddress`
    - foregroundColor: White
    - textSize: **14**
    - Text style: Bold
    - **Contents**: `CC Address`
5. Drag a new label to the card and set the following properties:
    - **Name**: `lbl_subject`
    - foregroundColor: White
    - textSize: **14**
    - Text style: Bold
    - **Contents**: `Subject`

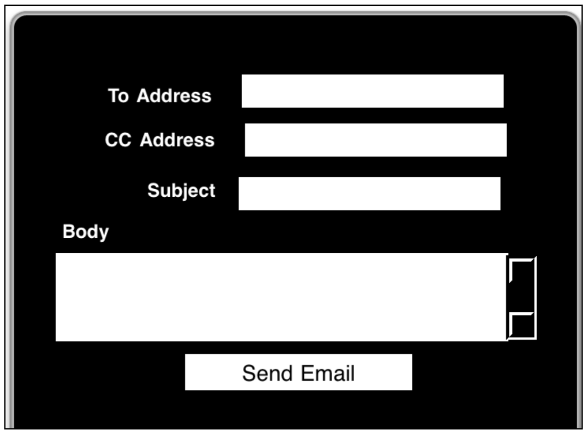
6. Drag a new label to the card and set the following properties:
    - **Name**: `lbl_body`
    - foregroundColor: White
    - textSize: **14**
    - Text style: Bold
    - **Contents**: `Body`
7. Drag a new **Text Entry Field** to the card, to the right of the `lbl_toAddress` label. Set the following properties:
    - **Name**: `fld_toAddress`
    - textSize: **14**
    - Text style: Bold
    - showBorder: Keep it unchecked
    - backgroundColor: White
8. Drag a new **Text Entry Field** to the card, to the right of the `lbl_ccAddress` label. Set the following properties:
    - **Name**: `fld_ccAddress`
    - textSize: **14**
    - Text style: Bold
    - showBorder: Keep it unchecked
    - backgroundColor: White
9. Drag a new **Text Entry Field** to the card, to the right of the `lbl_subject` label. Set the following properties:
    - **Name**: `fld_subject`
    - textSize: **14**
    - Text style: Bold
    - showBorder: Keep it unchecked
    - backgroundColor: White
10. Drag a new **Scrolling Field** to the card, just under the `lbl_body` label. Name the field `fld_body`.
11. Drag a new button to the card and set the following properties:
    - **Name**: `btn_sendEmail`
    - **Label**: `Send Email`
    - **Width**: 144

- **threeD**: Keep it unchecked
- **showBorder**: Keep it unchecked
- **hiliteBorder**: Keep it unchecked
- backgroundColor: White

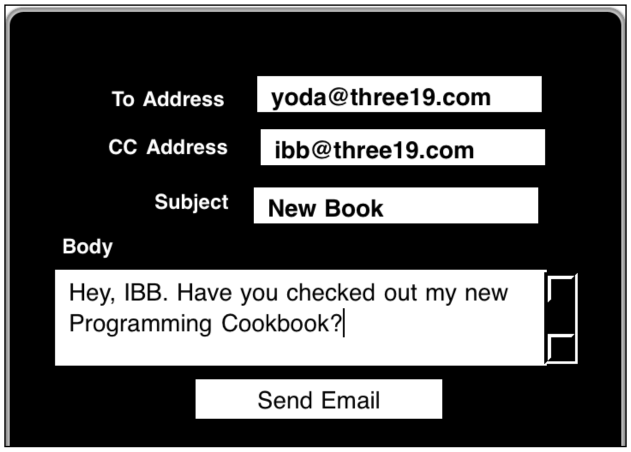
12. Add the following code to the `btn_sendEmail` button:

```
on mouseUp
  local tAddress, tCC, tSubject, tBody

  put the text of fld "fld_toAddress" into tAddress
  put the text of fld "fld_ccAddress" into tCC
  put the text of fld "fld_subject" into tSubject
  put the text of fld "fld_body" into tBody

  revMail tAddress, tCC, tSubject, tBody
end mouseUp
```

13. Align your objects so that the layout is similar to what is illustrated in the following screenshot:



14. Run the app in the simulator and add information in the text input fields, as shown in the following screenshot:


To Address: yoda@three19.com

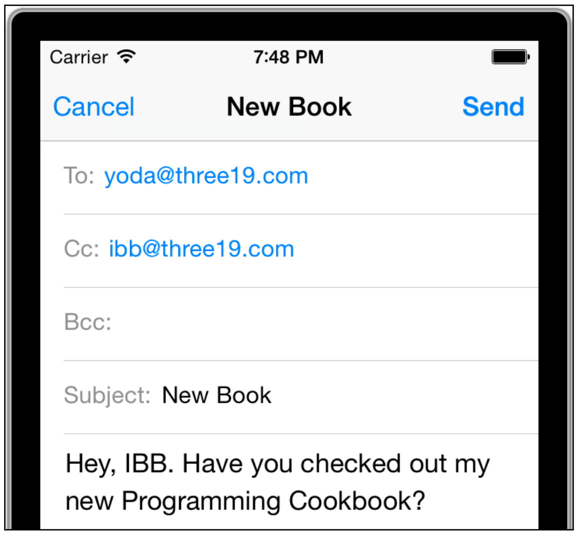
CC Address: ibb@three19.com

Subject: New Book

Body

Hey, IBB. Have you checked out my new Programming Cookbook?

15. Click on the **Send Email** button and the mobile device's e-mail client should launch and be prepopulated with the data you entered, as illustrated in the following screenshot:



To: yoda@three19.com

Cc: ibb@three19.com

Bcc:

Subject: New Book

Hey, IBB. Have you checked out my new Programming Cookbook?

How it works...

LiveCode makes our work easy when it comes to invoking the e-mail client. We made a call to the `revMail` command and the e-mail client's **New Message** dialog was instantiated. We passed four parameters to `revMail`: the To e-mail address, CC e-mail address, e-mail subject, and e-mail body.

See also

- The *Sending an e-mail* recipe

6
Data Structures

In this chapter, we will cover the following recipes:

- Using arrays
- Using multidimensional arrays
- Saving external data
- Loading external data
- Reading XML
- Writing XML
- Using SQLite
- Using MySQL

Introduction

In this chapter, you will learn how to accomplish tasks related to working data that are both in memory and external to your mobile app. External data can be very important to apps and can be structured in many different ways. The recipes in this chapter address data structured in arrays, pure text, XML, and databases.

Using arrays

Arrays are sets of data that are arranged in a specific pattern. A one-dimension array can be viewed as an ordered list of items. For example, we might have a list of players on our sports team, and can illustrate this as a simple list, as shown:

1. Peter Sebastian
2. Sol Gladman
3. Jonas Mathling

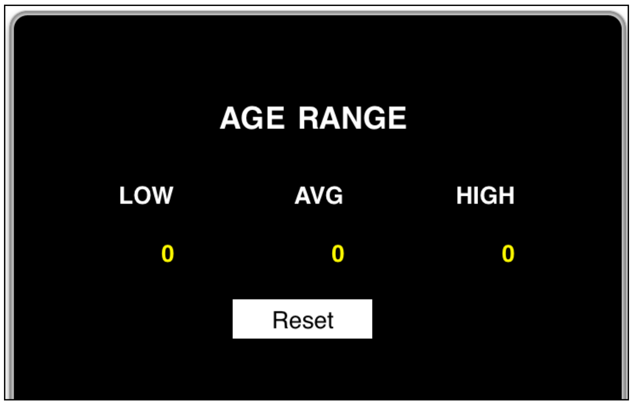
In our previous example, position 2 in our array is "Sol Gladman". This recipe will teach you how to create and use a one-dimensional array.

How to do it...

Perform the following steps to create and use a one-dimensional array:

1. Create a new main stack in LiveCode.
2. Set the background color of the default card to black.
3. Drag a new label to the card and set the following properties:
    - **Name**: `lbl_title`
    - foregroundColor: White
    - **Width**: `130`
    - **Height**: `30`
    - Size of text: `18`
    - Select bold
    - **Contents**: `AGE RANGE`
4. Drag a new label to the card and set the following properties:
    - **Name**: `lbl_low`
    - foregroundColor: White
    - **Width**: `50`
    - **Height**: `21`
    - Size of text: `14`
    - Select bold
    - **Contents**: `LOW`
5. Make a copy of the `lbl_low` label and change the following properties:
    - **Name**: `lbl_avg`
    - **Contents**: `AVG`
6. Make a copy of the `lbl_low` label and change the following properties:
    - **Name**: `lbl_high`
    - **Contents**: `HIGH`
7. Make a copy of the `lbl_low` label and change the following properties:
    - **Name**: `lbl_lowValue`
    - foregroundColor: Yellow
    - **Contents**: `0`

8. Make a copy of the `lbl_lowValue` label and change the name of the new label to `lbl_avgValue`.
9. Make a copy of the `lbl_lowValue` label and change the name of the new label to `highValue`.
10. Drag a new button to the card and set the following properties:
    - **Name**: `btn_reset`
    - **Label**: `Reset`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - hiliteBorder: Keep it unchecked
    - backgroundColor: White
11. Align your objects so that the layout is similar to what is illustrated in the following screenshot:



12. Add the following code to the `btn_reset` button to set up local variables:

```
on mouseUp
  local aAges, tLen, tKeys
  local aSortedAges, tIndex, tIncrement
  local tAvg
end mouseUp
```

13. Add the following additional code to the `btn_reset` button to clear the onscreen values:

```
  set the text of fld "lbl_lowValue" to 0
  set the text of fld "lbl_avgValue" to 0
  set the text of fld "lbl_highValue" to 0
```

14. Add the following additional code to the `btn_reset` button to determine the length of the array:

    ```
    put the number of lines of keys of aAges into tLen
    ```

15. Add the following additional code to the `btn_reset` button to sort the array numerically:

    ```
    get the keys of aAges
    sort lines of it by aAges[each]
    split it by return
    put it into tIndex
    ```

16. Add the following additional code to the `btn_reset` button to create a new array with the sorted results:

    ```
    put 1 into tIncrement
    repeat for each element tIndex in it
      put aAges[tIndex] into aSortedAges[tIncrement]
      add 1 to tIncrement
    end repeat
    ```

17. Add the following additional code to the `btn_reset` button to determine the average age of those listed in the array:

    ```
    put 0 into tAvg
    repeat with x = 1 to tLen
      add aSortedAges[x] to tAvg
    end repeat
    put tAvg / tLen into tAvg
    ```

18. Add the following additional code to the `btn_reset` button to display the results on the screen:

    ```
    set the text of fld "lbl_lowValue" to aSortedAges[1]
    set the text of fld "lbl_avgValue" to tAvg
    set the text of fld "lbl_highValue" to aSortedAges[tLen]
    ```

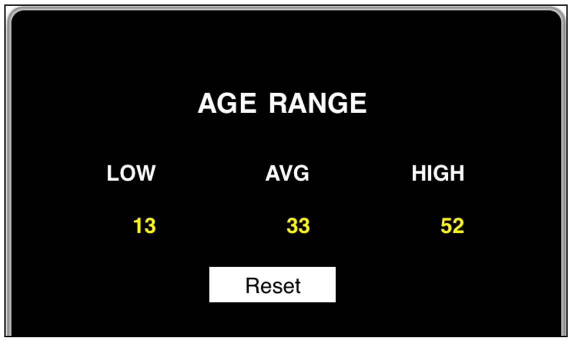
19. Ensure that your final code matches what is provided next:

> This code has several comments (preceded by the # symbol) that help organize and make sense of the code.

```
on mouseUp
  local aAges, tLen, tKeys
  local aSortedAges, tIndex, tIncrement
  local tAvg

  # clear on-screen values
```

```
  set the text of fld "lbl_lowValue" to 0
  set the text of fld "lbl_avgValue" to 0
  set the text of fld "lbl_highValue" to 0

  # populate the array
  put 25 into aAges[1]
  put 44 into aAges[2]
  put 13 into aAges[3]
  put 33 into aAges[4]
  put 42 into aAges[5]
  put 52 into aAges[6]
  put 34 into aAges[7]
  put 22 into aAges[8]
  put 32 into aAges[9]

  # determine length of array
  put the number of lines of keys of aAges into tLen

  # sort the array numerically
  get the keys of aAges
  sort lines of it by aAges[each]
  split it by return
  put it into tIndex

  # create a new array for the sorted results
  put 1 into tIncrement
  repeat for each element tIndex in it
    put aAges[tIndex] into aSortedAges[tIncrement]
    add 1 to tIncrement
  end repeat

  # get avg value
  put 0 into tAvg
  repeat with x = 1 to tLen
    add aSortedAges[x] to tAvg
  end repeat
  put tAvg / tLen into tAvg

  # update screen values
  set the text of fld "lbl_lowValue" to aSortedAges[1]
  set the text of fld "lbl_avgValue" to tAvg
  set the text of fld "lbl_highValue" to aSortedAges[tLen]
end mouseUp
```

20. Run your app in the mobile simulator.
21. Click on or tap the **btn_reset** button. You should see results that are identical to what is illustrated in the following screenshot:



How it works...

We experimented with one-dimensional arrays by first creating and populating an array of numeric values (ages). We calculated the number of values and sorted the array numerically. We took the newly sorted information and populated a new, sorted array. When we have large data in the memory, it can be much quicker to access that information if it is sorted in a manner that is relevant to our app. We next retrieved specific values for onscreen display.

See also

- The *Using multidimensional arrays* recipe

Using multidimensional arrays

One-dimensional arrays are simple lists of information. Multidimensional arrays can hold related data in columns and rows such as with a spreadsheet. This recipe will show you how to create, populate, and retrieve information from a multidimensional array.

How to do it...

Perform the following steps to create and use multidimensional arrays:

1. Create a new main stack in LiveCode.
2. On the stack or card, create the following code to create a two-dimensional array and populate it with three rows of information:

```
command makemyarray
  local aMyArray
```

```
    put "J" into aMyArray[1][1]
    put "Jones" into aMyArray[1][2]
    put "United States" into aMyArray[1][3]
    --
    put "R" into aMyArray[2][1]
    put "Smith" into aMyArray[2][2]
    put "United Kingdom" into aMyArray[2][3]
    --
    put "T" into aMyArray[3][1]
    put "Johnson" into aMyArray[3][2]
    put "Canada" into aMyArray[3][3]

end makemyarray
```

3. Use the following line of code to retrieve a specific piece of information. In this case, we are looking for the third column of the third row:

```
put aMyArray[3][3]
```

> The previous line of code can be placed inside the `makeMyArray` code, preceding the end `makeMyArray` statement. For testing in the LiveCode IDE, you can simply type the line of code in the message box to obtain the results.

How it works...

To create a multidimensional array, we envision our data as being in a matrix such as a spreadsheet. If we have two columns, then each cell will be viewed as `myArray[1][1]` and `myArray[1][2]`. We can target specific cells such as `myArray[5][4]` for the fifth row and fourth column.

There's more...

Sometimes, it can be helpful to evaluate a variable to determine whether it is an array or not. To do this, we can use the simple `put aMyArray is an array` command. This statement will be evaluated as `true` or `false`.

See also

- The *Using arrays* recipe

Saving external data

There are many reasons you might want your app to be able to save information on a mobile device such as saving scores, game progress, settings, and more. This recipe demonstrates how to save data to a mobile device. While this specific example is for iOS devices, the same principles apply to Android-based devices.

Getting ready

Writing files on physical devices is more complex with mobile devices as opposed to desktop applications. All files are confined to the app's home folder or subordinate folder. Here are the initial folders that are created with a standalone mobile app:

Folder	**Use**	**Sync'd by iTunes**
`cache`	Transient data (preserved between app launches)	No
`documents`	Documents	Yes
`engine`	App's binary files and bundled resources	Yes
`home`	The `home` folder for the app and supporting files	Yes
`temporary`	Data not needed in between app launches	No

So, before you get started, you need to know where you will save your external files.

How to do it...

Follow the steps in this recipe to create a `tryit.txt` file and save it to your mobile device's filesystem:

1. Create a new main stack in LiveCode.
2. Set the background color of the main card to black.
3. Drag a new button to the main card.
4. Add the following code to the new button:

```
on mouseUp
  set the defaultFolder to specialFolderPath("documents")

  put "This is a test" into URL("file:tryit.txt")
end mouseUp
```

How it works...

We changed our default folder to the `documents` folder with the special folder path. This points the system to the `documents` folder, which is a subordinate of our `app` folder on the mobile device. Also, if the `tryit.txt` file already existed, our code would have overwritten it.

See also

- The *Loading external data* recipe

Loading external data

There are many reasons you might want your app to be able to retrieve information from a mobile device, such as reading scores, game progress, settings, and more. This recipe demonstrates how to read data that was previously saved to a mobile device. While this specific example is for iOS devices, the same principles apply to Android-based devices.

Getting ready

This recipe references a file named `tryit.txt`, which was created with the *Saving external data* recipe. You will need to use this recipe prior to using the following one.

How to do it...

Use the following steps to read a file from your mobile device:

1. Create a new main stack in LiveCode.
2. Set the background color of the main card to black.
3. Drag a new button to the main card and set the following properties:
   - backgroundColor: White
   - showBorder: Keep it unchecked
   - hiliteBorder: Keep it unchecked
   - threeD: Keep it unchecked
4. Drag a new label to the main card and name it `results`.
5. Add the following code to the new button:

```
on mouseUp
  set the defaultFolder to specialFolderPath("documents")

  put URL ("file:tryit.txt") into field "results"
end mouseUp
```

When reading information from an external file, you can put the results directly to the screen via a pop up, in a field, or you can put it into memory for further manipulation. In the example of this recipe, the results are echoed to the screen via the `results` field.

How it works...

We changed our default folder to the `documents` folder with the special folder path. This points the system to the `documents` folder, which is a subordinate of our `app` folder on the mobile device.

See also

- The *Saving external data* recipe

Reading XML

XML (**Extensible Markup Language**) files represent an organized method to save and quickly read information from files.

How to do it...

Use the following steps to read the XML data:

1. Create a new main stack in LiveCode.
2. Set the background color of the main card to black.
3. Drag a new button to the card and assign the following properties:
   - **Name**: `btn_readXML`
   - **Label**: `Read XML`
   - threeD: Keep it unchecked
   - showBorder: Keep it unchecked
   - hiliteBorder: Keep it unchecked
   - backgroundColor: White
4. Create a text file with the following text. Save the file as `sample.xml` into the `documents` folder on your mobile device:

```
<sample>
  <language>English</language>
  <timeZone>ET</timeZone>
</sample>
```

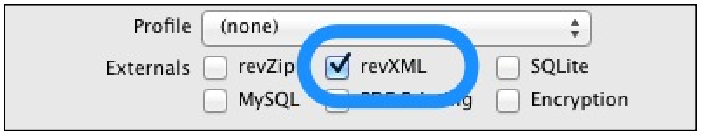
5. Assign the following code to the `btn_readXML` button:

```
on mouseUp
  local xmlTree, tLanguage

  # Read XML File into memory
  put readXMLtoTree() into xmlTree

  # Retrieve specific piece of information
  put revXMLNodeContents(xmlTree, "sample/language") into
    tLanguage

  # Release memory
  revDeleteXMLTree xmlTree
end mouseUp

private function readXMLtoTree
  local xmlFile, xmlData, xmlTree

  answer file "Select your XML file"
  if the result is not "Cancel" then
    put it into xmlFile
  end if
  put url ("file:" & xmlFile) into xmlData

  # new XML tree
  put revCreateXMLTree(xmlData, false, true, false) into
    xmlTree

  return xmlTree
end readXMLtoTree
```

[The `answer file` function is not available for mobile operating systems currently. It is used in this recipe for illustrative purposes.]

6. Open the Standalone Application Settings dialog box and click on the **iOS** tab.

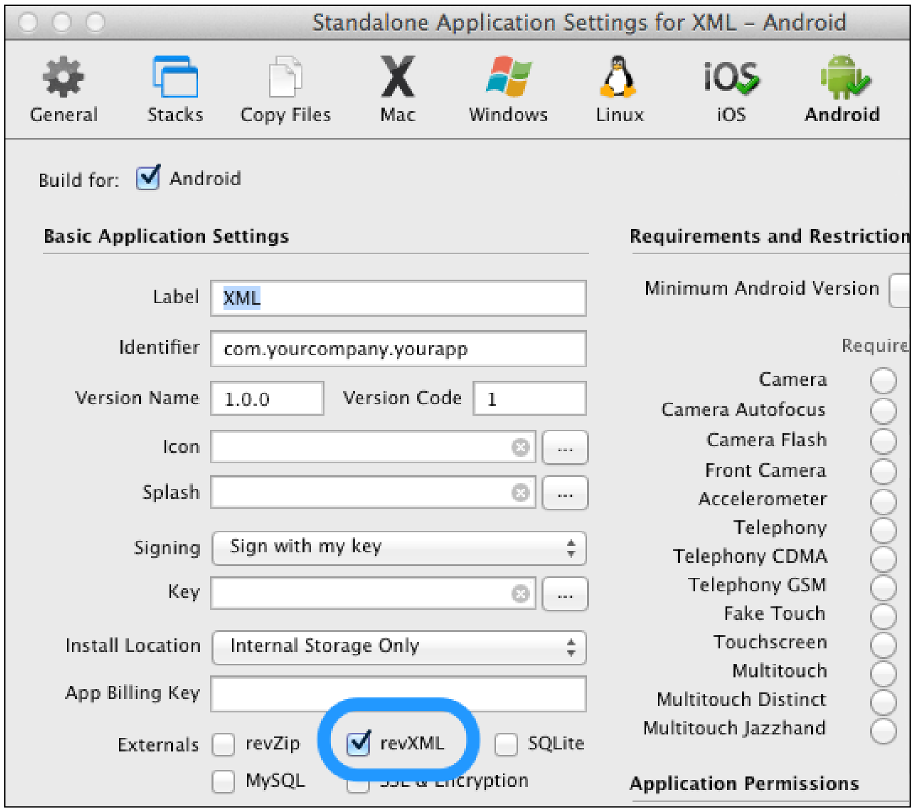
7. Check the **revXML** checkbox by navigating to **Basic Application Settings** | **Externals**. See the following screenshot for reference:



Profile (none)

Externals [ ] revZip [✓] revXML [ ] SQLite [ ] MySQL [ ] Encryption

8. Open the Standalone Application Settings dialog box and click on the **Android** tab.
9. Check the **revXML** checkbox by navigating to **Basic Application Settings** | **Externals**. See the following screenshot for reference:



Build for: [✓] Android

Basic Application Settings

Label	XML
Identifier	com.yourcompany.yourapp
Version Name	1.0.0
Version Code	1
Icon	
Splash	
Signing	Sign with my key
Key	
Install Location	Internal Storage Only
App Billing Key	
Externals	[ ] revZip [✓] revXML [ ] SQLite [ ] MySQL [ ] SSL & Encryption

How it works...

This recipe illustrated how to read from an XML file. We created a private function called `readXMLtoTree` that retrieves the XML file and populates a new XML tree in the memory. We then used `put revXMLNodeContents(xmlTree, "sample/language")` in the `tLanguage` statement to extract the language out of the XML tree.

See also

- The *Writing XML* recipe

Writing XML

XML files represent an organized method to save and quickly read information from files.

Getting ready

Since XML is a marked language used for organizing and nesting data, you must first determine what type of data you have and how you want it organized. For example, you might simply have vehicle makes and models, so your XML file structure would be `<vehicle><make><model>` and might look something like the following code:

```
<vehicle>
  <ford>
    <car>Taurus</car>
    <truck>F-350</truck>
    <van>E-150</van>
  </ford>
```

How to do it...

Use the following steps to create and write an XML data structure:

1. To write XML data, use a function to write each line of text such as with the following code:

   ```
   function createXML theTag, theText
     local theXML

     put "<" & theTag & ">" into theXML
     put theText after theXML
     put "</" & theTag & ">" after theXML
     return theXML
   end createXML
   ```

2. Once the entire XML file is constructed in the memory, you can save it to a physical storage medium (see the *Saving external data* recipe for details).

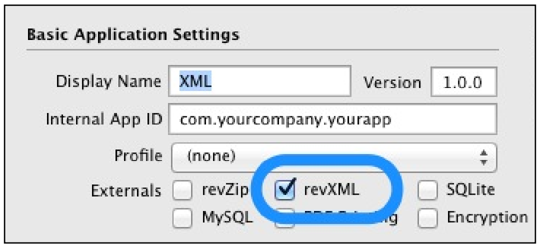
How it works...

XML is a tagged language, so we can easily add the `<value>` and `</value>` open and closing tags to text values. While the fundamentals of XML are easily understood, the implementation of XML is a bit more complex and requires precise coding.

There's more...

This recipe demonstrated how to write XML data without using any of LiveCode's built-in XML commands. If you use these built-in commands, then you will want to ensure that you select **revXML** by navigating to **Basic Application Settings** | **Externals** under the Standalone Application Settings dialog box.

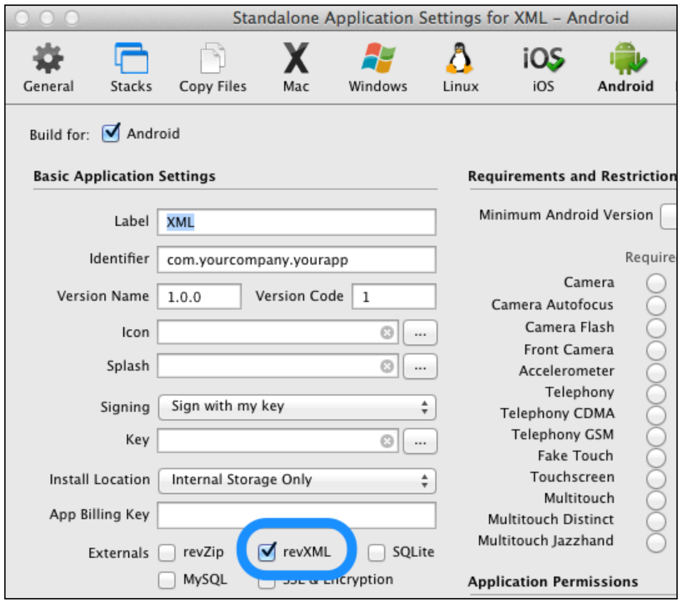
Take a look at the following screenshot for iOS:



Basic Application Settings

Display Name: XML
Version: 1.0.0
Internal App ID: com.yourcompany.yourapp
Profile: (none)
Externals: ☐ revZip ☑ revXML ☐ SQLite ☐ MySQL ☐ Encryption

Take a look at the following screenshot for Android:



Build for: ☑ Android

Basic Application Settings

Label: XML
Identifier: com.yourcompany.yourapp
Version Name: 1.0.0
Version Code: 1
Icon:
Splash:
Signing: Sign with my key
Key:
Install Location: Internal Storage Only
App Billing Key:
Externals: ☐ revZip ☑ revXML ☐ SQLite ☐ MySQL ☐ SSL & Encryption

Requirements and Restriction

Minimum Android Version

Require

Camera ○
Camera Autofocus ○
Camera Flash ○
Front Camera ○
Accelerometer ○
Telephony ○
Telephony CDMA ○
Telephony GSM ○
Fake Touch ○
Touchscreen ○
Multitouch ○
Multitouch Distinct ○
Multitouch Jazzhand ○

Application Permissions

See also

- The *Reading XML* recipe

Using SQLite

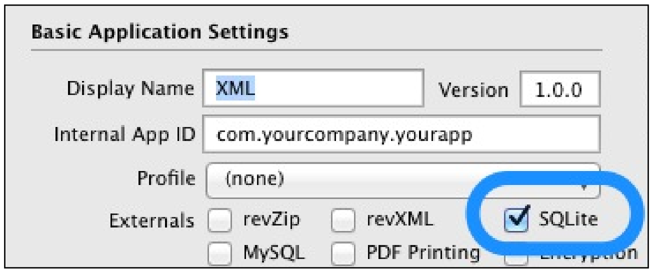
Databases are advanced data repositories known as relational database management systems. SQLite is an embedded database that can be used in your LiveCode mobile apps, giving you tremendous capability to store, retrieve, and manipulate data. Unlike MySQL databases, SQLite databases do not require a server. In this recipe, we will create a mobile app that uses five SQLite functions:

1. Connect to a database.
2. Add a table to the database.
3. Add data to the database table.
4. Retrieve data from the database.
5. Close the database.

How to do it...

Use the following steps to create a mobile app that instantiates a SQLite database and add and retrieve information from it:

1. Create a new main stack in LiveCode.
2. Set the background color of the main card to black.
3. Open the Standalone Application Settings dialog box and click on the **iOS** tab.
4. Check the **SQLite** checkbox by navigating to **Basic Application Settings** | **Externals**. See the following screenshot for reference:

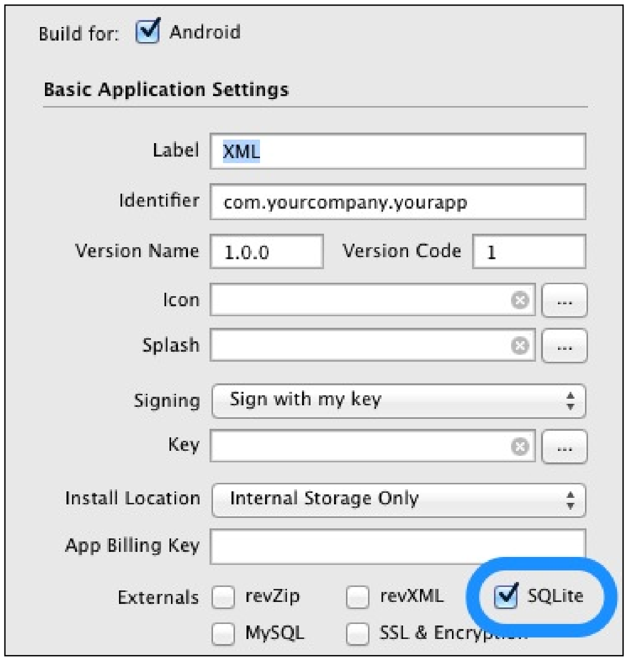


Basic Application Settings

Display Name: XML
Version: 1.0.0
Internal App ID: com.yourcompany.yourapp
Profile: (none)
Externals: ☐ revZip ☐ revXML ☑ SQLite ☐ MySQL ☐ PDF Printing ☐ Encryption

5. Open the Standalone Application Settings dialog box and click on the **Android** tab.

6. Check the **SQLite** checkbox by navigating to **Basic Application Settings** | **Externals**. See the following screenshot for reference:



Build for: [✓ Android]

Basic Application Settings

Label	XML
Identifier	com.yourcompany.yourapp
Version Name	1.0.0
Version Code	1
Icon	
Splash	
Signing	Sign with my key
Key	
Install Location	Internal Storage Only
App Billing Key	
Externals	revZip, revXML, [✓ SQLite], MySQL, SSL & Encryption

7. Drag a new button to the card and assign the following properties:
    - **Name**: `btn_Connect`
    - **Label**: `1 – Connect`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - hiliteBorder: Keep it unchecked
    - backgroundColor: White
    - **Width**: `130`
    - lockLoc: Keep it checked

8. Drag a second button to the card and assign the following properties:
    - **Name**: `btn_AddTable`
    - **Label**: `2 – Add Table`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked

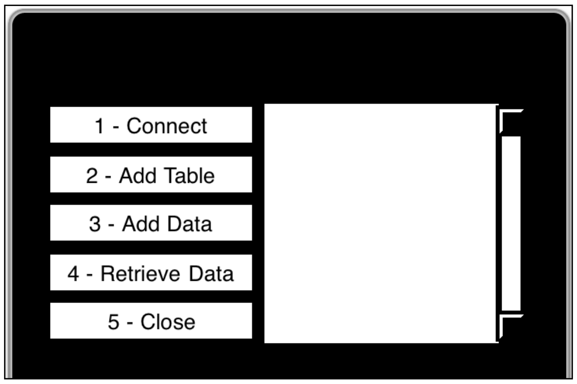
- hiliteBorder: Keep it unchecked
- backgroundColor: White
- **Width**: `130`
- lockLoc: Keep it checked

9. Drag a third button to the card and assign the following properties:
    - **Name**: `btn_AddData`
    - **Label**: `3 – Add Data`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - hiliteBorder: Keep it unchecked
    - backgroundColor: White
    - **Width**: `130`
    - lockLoc: Keep it checked

10. Drag a fourth button to the card and assign the following properties:
    - **Name**: `btn_RetrieveData`
    - **Label**: `4 – Retrieve Data`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - hiliteBorder: Keep it unchecked
    - backgroundColor: White
    - **Width**: `130`
    - lockLoc: Keep it checked

11. Drag a fifth button to the card and assign the following properties:
    - **Name**: `btn_Close`
    - **Label**: 5 – `Close`
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - hiliteBorder: Keep it unchecked
    - backgroundColor: White
    - Width: `130`
    - lockLoc: Keep it checked

12. Drag a scrolling field to the card and assign the following properties:
    - **Name**: `fld_output`
    - **Width**: `170`
    - **Height**: `154`
    - traversalOn: Keep it unchecked
    - showFocusBorder: Keep it unchecked
    - threeD: Keep it unchecked
    - showBorder: Keep it unchecked
    - backgroundColor: White
    - lockLoc: Keep it checked
13. Align the six onscreen objects so that they are laid out as illustrated in the following screenshot:



14. Next, we need to connect to the database. If the database does not already exist, then we will create one as part of the connection process. Add the following code to the `btn_Connect` button:

```
on mouseUp
  global dbID
  local dbPath

  put specialFolderPath("documents") & \
    "/packtcookbook.sqlite" into dbPath
  put revOpenDatabase("sqlite", dbPath,,,,) \
    into dbID

  put "Database ID " & dbID & " created" \
    into fld "fld_output"
  set the enabled of me to false
end mouseUp
```

15. Now, let's code the fifth button so that we can have the ability to close an open database. This will free up the system memory. Add the following code to the `btn_Close` button:

```
on mouseUp
  global dbID

  revCloseDatabase dbID

  put "Database ID " & dbID & " closed" \
    into fld "fld_output"
  set the enabled of btn "btn_Connect" to true
end mouseUp
```

16. Next, we will add a functionality to add a table to the database. We will add a table to hold the account information. Add the following code to the `btn_AddTable` button:

```
on mouseUp
  global dbID
  local tmpSQL

  put "CREATE TABLE account_data (bank char(20),
    password char(10))" into tmpSQL
  revExecuteSQL dbID, tmpSQL

  put quote & "account_data" & quote & \
    " table created" into fld "fld_output"
end mouseUp
```

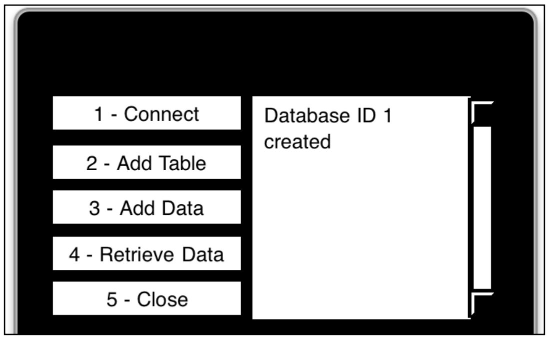
17. We are now ready to add data to our table. We'll add information regarding three bank accounts. Add the following code to the `btn_AddData` button:

```
on mouseUp
  global dbID
  local tmpSQL

  put "INSERT into account_data VALUES ('19th
    Bank','99TA4509');" into tmpSQL
  put "INSERT into account_data VALUES ('Bank
    of the Poor','3309942');" after tmpSQL
  put "INSERT into account_data VALUES ('Blue
    Feet Bank','0000445');" after tmpSQL
  revExecuteSQL dbID, tmpSQL

  put "Three records added." into fld "fld_output"
end mouseUp
```

18. Our last task is to query the database and display the results. Add the following code to the `btn_RetrieveData` button:

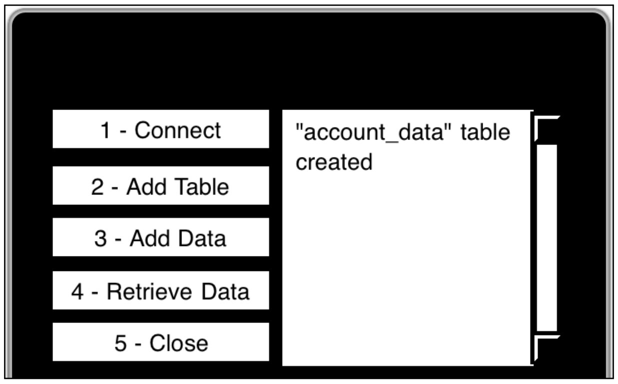
```
on mouseUp
  global dbID
  local theRecords, tmpSQL

  put "SELECT * from account_data" into tmpSQL
  put revDataFromQuery(tab,return,dbID,tmpSQL) \
    into theRecords
  put theRecords into fld "fld_output"
end mouseUp
```

19. Now, you are ready to test the application in a mobile simulator or on your actual device. Run the application in a simulator (or an actual device).
20. Click on the button labeled 1 – Connect. You should see the output reflected in the following screenshot:



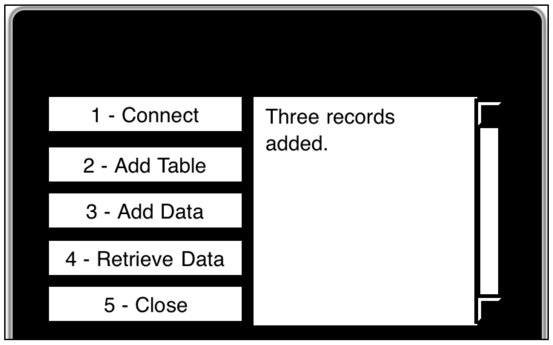
Database ID 1 created

21. Click on the button labeled 2 – Add Table. You should see the output reflected in the following screenshot:



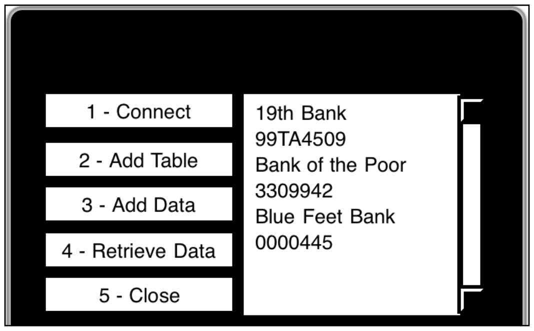
"account_data" table created

22. Click on the button labeled 3 – Add Data. You should see the output reflected in the following screenshot:



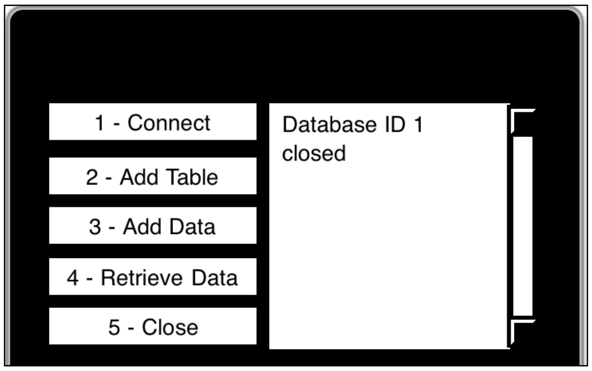
```
Three records
added.
```

23. Click on the button labeled 4 – Retrieve Data. You should see the output reflected in the following screenshot:



```
19th Bank
99TA4509
Bank of the Poor
3309942
Blue Feet Bank
0000445
```

24. Click on the button labeled 5 – Close. You should see the output reflected in the following screenshot:



```
Database ID 1
closed
```

How it works...

We took a five-step process towards using SQLite in LiveCode. We started by creating and connecting to the database. Next, we added a table and then some data. Then, we retrieved and displayed the database data. Lastly, we closed the database.

We used several commands built into the LiveCode engine and ensured that we would be using the SQLite external.

There's more...

There are some excellent SQLite IDEs available, such as SQLite Expert, SQLite Designer, SQLite Administrator, SQLite Database Browser, and SQLiteSpy, that can be used to create and edit a SQLite database. It is recommended that you design and populate your SQLite database using one of these or another tool and then create your LiveCode mobile app to interface with that database. This approach saves you the time required to script your own SQLite functionality.

See also

- The *Using MySQL* recipe
- The *Loading external data* recipe
- The *Saving external data* recipe

Using MySQL

MySQL databases are relational data management systems that require a server to reside in. We interface with these databases with user accounts established by the database administrator. We use MySQL syntax, which is a specific query language, to read, write, and modify the database. In this recipe, you will learn how to connect to a MySQL database, how to read it, and how to modify it.

Getting ready

You will need to know the specific address of your database, the username, the password, and the port number.

How to do it...

Use the following information to integrate the MySQL database interaction into your mobile app:

1. To connect to a MySQL database, we use the following code. Be sure to replace each ______ with your actual data:

   ```
   global dbID
   local dbAddress, dbName, dbUser, dbPassword

   put "_____" into dbAddress
   put "_____" into dbName
   put "_____" into dbUser
   put "_____" into dbPassword

   put revOpenDatabase("MySQL", dbAddress, dbName, dbUser,
     dbPassword) into dbID
   ```

2. Our second major function is to retrieve information from the database. We will need to know what tables your database has. Use the following code to query the MySQL database to retrieve all the data from a specific table. Be sure to replace ______ with your actual data. The results are housed in the `theTableData` local variable, allowing you to manipulate or display the data:

   ```
   global dbID
   local dbTableName, tmpSQL, theTableData

   put "_____" into dbTableName
   put "SELECT * FROM " & dbTableName into tmpSQL
   put revDataFromQuery(tab, cr, dbID, tmpSQL) into
     theTableData
   ```

3. Next, we will add a new record to the database. We require two items to accomplish this task. We need a properly formed SQL query, and we must use the `revExecuteSQL` command. Use the following code to add a record to your database. Be sure to replace ______ with your actual data. This example assumes that the table consists of `fName` and `lname` fields for the first and last names:

   ```
   global dbID
   local dbTableName, tmpSQL
   local tmpFields, tmpFName, tmpLName

   put "_____" into dbTableName
   put "fname, lname" into tmpFields
   put "Happy" into tmpFName
   put "Jones" into tmpLName
   ```

```
put "INSERT INTO " & dbTableName & " (" & tmpFields & ) VALUES
(:1, :2) into tmpSQL
revExecuteSQL dbID, tmpSQL, "tmpFName", "tmpLName"
```

4. To edit a record, you simply write a SQL statement using the `UPDATE` command. Then, you use the `revExecuteSQL` LiveCode command.
5. To delete a record, you simply write a SQL statement using the `DELETE FROM` command. Then, you use the `revExecuteSQL` LiveCode command.
6. The last thing we need to do is to disconnect from the database. We do this with the following lines of code:

```
global dbID
revCloseDatabase dbID
```

How it works...

There are two key factors to using a MySQL database with LiveCode. First, you must have precise information regarding the location (address), port, and user credentials (username and password). Secondly, you must be able to write accurate SQL statements. There are ample books and Internet resources that will help you learn the **Structured Query Language** (**SQL**).

See also

- The *Using SQLite* recipe

7
External Media

In this chapter, we will cover the following recipes:

- Loading an image
- Capturing an image from a mobile device's camera
- Resizing an image
- Playing a movie
- Controlling the movie playback
- Playing an audio file

Introduction

In this chapter, you will learn how to accomplish tasks related to working with external media for your iOS and Android apps. You will learn how to load, display, and resize images. You will learn how to play a movie and control playback by creating custom controls. You will also learn how to play audio files from within your mobile apps.

Loading an image

Loading an image to a LiveCode stack is straightforward. In this recipe, you will learn how to load an image from the mobile device's photo library.

How to do it...

This recipe provides details on how to load an image from a mobile device's library.

Use the following code to prompt the user to select an image from their device's image library:

```
on mouseUp
  mobilePickPhoto "library"
  if the result is empty then
    // do something
  end if
end mouseUp
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

How it works...

For both iOS and Android devices, we can use the `mobilePickPhoto` command to prompt the user to select an image from their device's image library. We accomplish this by passing `library` as the parameter. Our code included a check to see whether the result was empty. In this case, we are checking to see whether the selected image was loaded by the `mobilePickPhoto` command.

There's more...

If you want to have the user select an image from the mobile device's camera roll, then you simply use `album` as the parameter instead of `library` when invoking the `mobilePickPhoto` command.

See also

- The *Capturing an image from a mobile device's camera* recipe
- The *Resizing an image* recipe

Capturing an image from a mobile device's camera

In this recipe, you will learn how to take a picture using the mobile device's camera.

How to do it...

Use the following code to prompt the user to select an image from their device's image library:

```
on mouseUp
  mobilePickPhoto "camera"
  if the result is empty then
    // do something
  end if
end mouseUp
```

How it works...

For both iOS and Android devices, we can use the `mobilePickPhoto` command to prompt the user to snap a photo using the device's camera. We accomplish this by passing `camera` as the parameter.

There's more...

For iOS devices only (not Android), we can use `rear camera` or `front camera` as the parameter to specify which camera is to be used. Also, as you might assume, we cannot test a mobile device's camera in the simulator.

See also

- The *Loading an image* recipe
- The *Resizing an image* recipe

Resizing an image

Resizing images can be a useful utility in some mobile apps. This is especially important if your app allows the user to capture or import images. This recipe will show you how to resize an image in a mobile app.

How to do it...

Follow the steps in this recipe to resize an image using the LiveCode script:

1. To adjust the height of an image, enter the following code:

   ```
   set the height of img "myPhoto" to 200
   ```

   Note, that `myPhoto` should be the name of your actual image. Also, the value `200` should be replaced with the actual height (in pixels) that you want the image to have.

2. To adjust the width of an image, enter the following code:

   ```
   set the width of img "myPhoto" to 250
   ```

   Note, that `myPhoto` should be the name of your actual image. Also, the value `250` should be replaced with the actual width (in pixels) that you want the image to have.

How it works...

LiveCode makes resizing images very easy. We simply need to set the `height` and `width` properties to have the image resized.

There's more...

When images are resized, it is a good idea to set the `lockLoc` parameter of the image to `true`. This will prevent the image from reverting to its original size. This can be accomplished with the following line of code:

```
set the lockLoc of img "myPhoto" to true
```

See also

- The *Loading an image* recipe
- The *Capturing an image from a mobile device's camera* recipe

Playing a movie

In this recipe, we will create a card that automatically plays an embedded movie when the card is loaded.

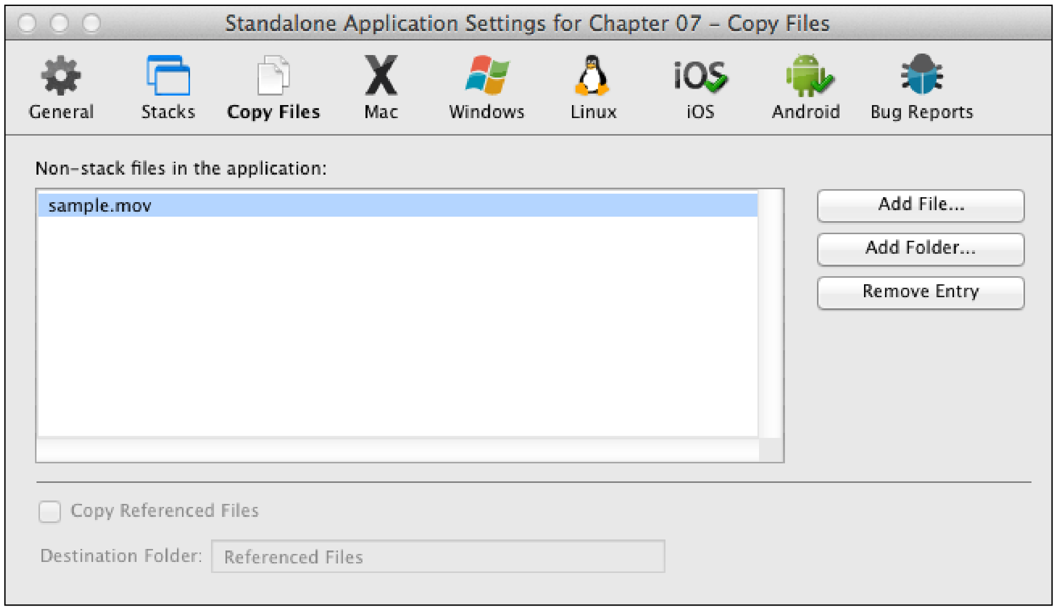
Getting ready

The `sample.mov` video file is provided on the book's website.

How to do it...

Follow the steps in this recipe to create the necessary controls that can play a movie using LiveCode:

1. Create a new main stack in LiveCode with the following properties:
   - **Height**: `360`
   - **Width**: `480`
   - backgroundColor: Black
2. Use the Standalone Application Settings dialog window and select the Copy Files icon.
3. Use the **Add Files** dialog box to upload the `sample.mov` file. Once this is done, your dialog window should look similar to the following screenshot:


Standalone Application Settings for Chapter 07 – Copy Files

Non-stack files in the application:

sample.mov

Copy Referenced Files

Destination Folder: Referenced Files

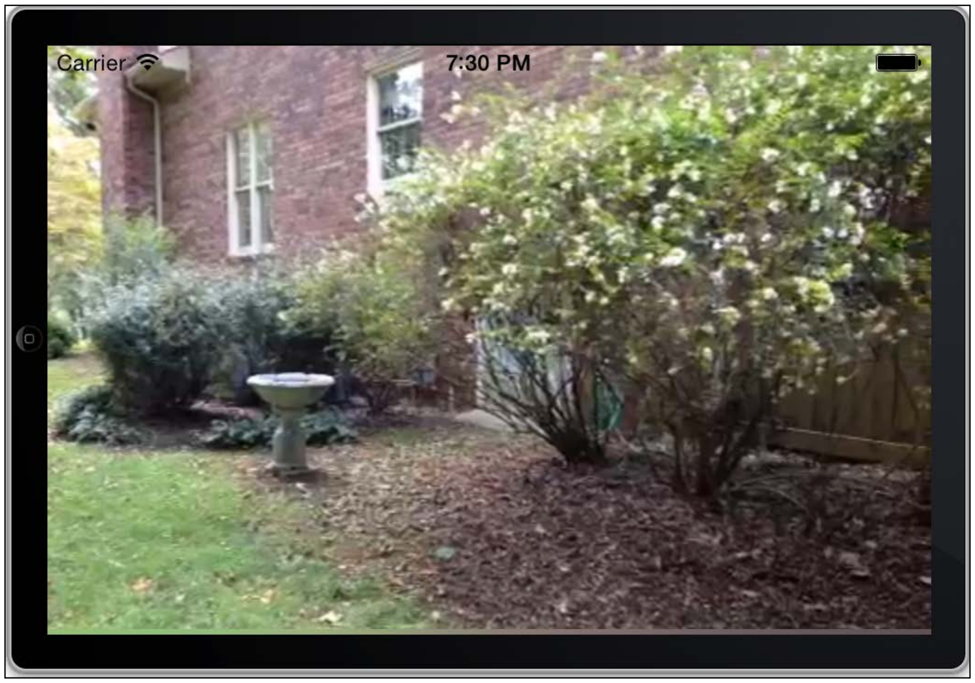
4. Close the Standalone Application Settings dialog window.
5. Add the following code to the card:

```
on openCard
  mobileControlCreate "player", "myController"

  mobileControlSet "myController", "filename",
    specialFolderPath("engine") & "/sample.mov"
  mobileControlSet "myController", "visible", true
  mobileControlSet "myController", "rect",
    "1,1,481,361"

  mobileControlDo "myController", "play"
end openCard
```

6. Add the following code to the card so the mobile control is deleted when the card is closed:

```
on closeCard
  mobileControlDelete "myController"
end closeCard
```

7. Test the app in the mobile simulator. You should see the video playing as shown:



How it works...

For iOS and Android mobile device apps, we use the `mobileControlSet`, `mobileControlCreate`, `mobileControlDelete`, `mobileControlDo`, and `mobileControlTarget` commands to display and control the video.

See also

- The *Controlling the movie playback* recipe

Controlling the movie playback

If your app contains video files and you want your users to be able to control certain aspects of it, then this recipe is for you. In this recipe, you will learn how to control the movie player on mobile devices.

How to do it...

Follow the given steps to play, pause, and stop a movie clip's playback in a mobile app written in LiveCode:

1. To start playing a video, use the following command:
   ```
   mobileControlDo "myController", "play"
   ```
2. To pause a video, use the following command:
   ```
   mobileControlDo "myController", "pause"
   ```
3. To stop playing a video, use the following command:
   ```
   mobileControlDo "myController", "stop"
   ```

For all of these steps, remember to replace `myController` with the name of the mobile controller you created using the `mobileControlCreate` command.

How it works...

To control a mobile movie controller, we make use of the `mobileControlCreate` command.

See also

- The *Playing a movie* recipe

Playing an audio file

In this recipe, we will create a card that automatically plays an embedded audio file when the card is loaded.

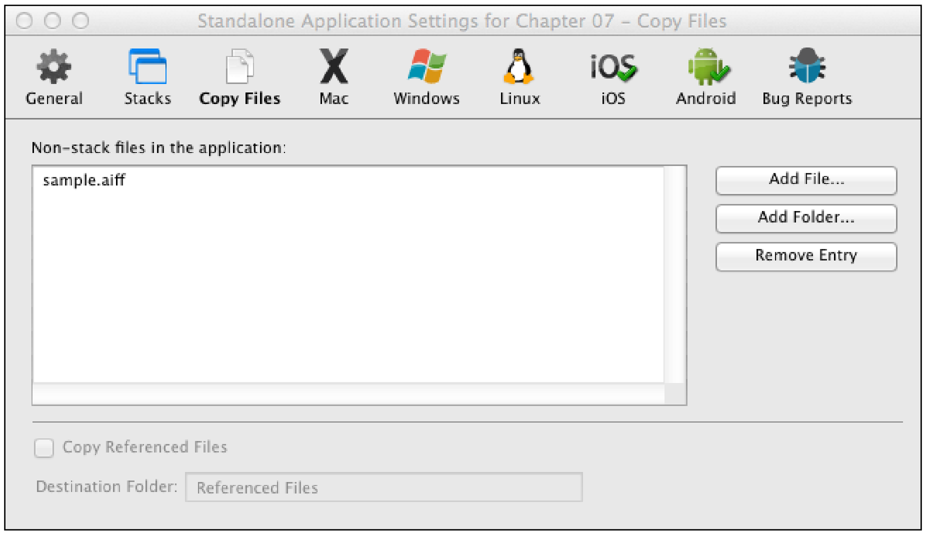
Getting ready

The `sample.aiff` audio file is provided on the book's website.

How to do it...

Use the following steps to load and play an audio file in a mobile app:

1. Create a new main stack in LiveCode with the following properties:
   - **Height**: `360`
   - **Width**: `480`
   - backgroundColor: Black
2. Use the Standalone Application Settings dialog window and select the Copy Files icon.
3. Use the **Add Files** dialog box to upload the `sample.aiff` file. Once this is done, your dialog window should look similar to the following screenshot:



Non-stack files in the application:

sample.aiff

Copy Referenced Files

Destination Folder: Referenced Files

4. Close the Standalone Application Settings dialog window.
5. Add the following code to the card:

```
on openCard
  play specialFolderPath("engine") & "/sample.aiff"
end openCard
```

6. Test the app in the mobile simulator. You should hear the audio playing.

How it works...

LiveCode allows us to use the `play` command to play embedded audio files on mobile devices. For this recipe, we made a single call to the audio file we added to our app via the **Copy Files** section of the Standalone Application Settings dialog window.

8
Using MobGUI

In this chapter, we will cover the following recipes:

- Setting up MobGUI
- Using a navigational bar
- Using a button
- Using a navigational button
- Using a slider
- Using a toggle button
- Using a list button
- Using a progress bar

Introduction

In this chapter, you will learn how to leverage the power of MobGUI, which is a LiveCode plugin, to accelerate your mobile app development. The MobGUI plugin is a commercial product that comes with a commercial or community LiveCode license. The plugin can be purchased from the LiveCode store. In addition, a free demo version can be downloaded at `www.mobgui.com/download.php`.

Setting up MobGUI

This recipe will walk you through the steps that are required to download and install the MobGUI plugin on your computer so that you can use it with all of your mobile app development projects using LiveCode.

Getting ready

Before starting with this recipe, you will need to decide whether you want to purchase the MobGUI plugin or simply download the demo version to experiment with. Once you make this decision, you will be ready for this recipe.

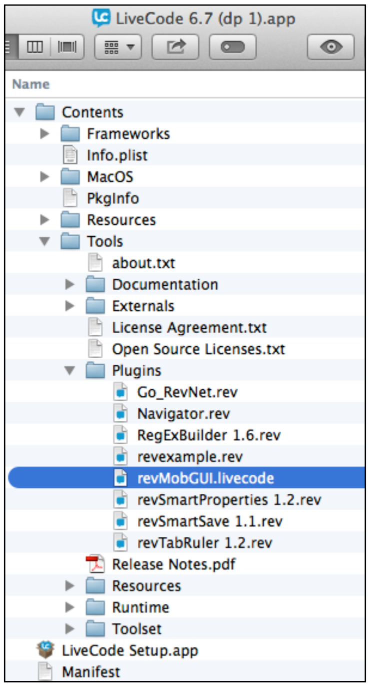
How to do it...

Follow this recipe's steps to download and install MobGUI on your development computer:

1. Download the MobGUI plugin from the LiveCode store (if purchasing it). If you are not making the purchase, then you can download the demo version directly from the MobGUI website. Select one of these options and download the file.
2. Unzip/decompress the downloaded file. This step will result in a single file named `revMobGUI.livecode`:

   If you are downloading the demo version, the filename will be `MobGUIDemo.livecode`.

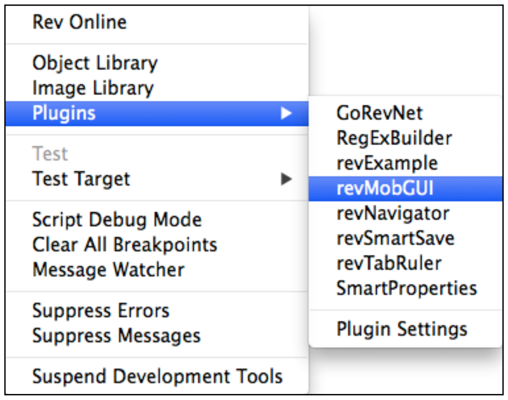
3. Drag the `revMobGUI.livecode` file to the `Plugins` folder/directory on your development computer as shown in the following screenshot:


LiveCode 6.7 (dp 1).app

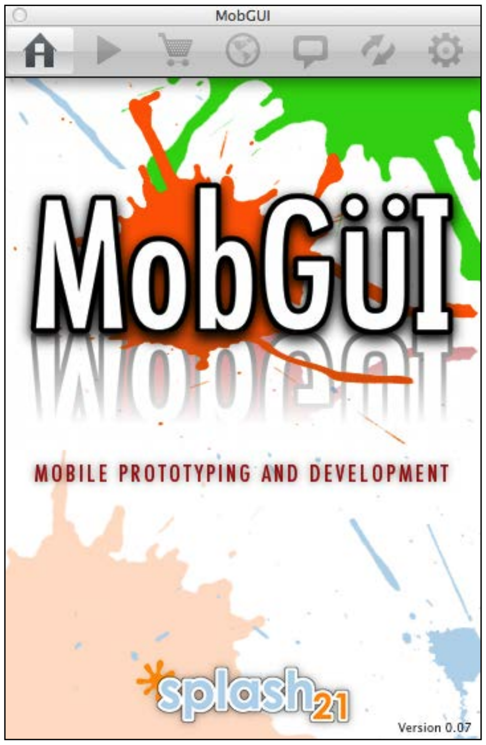
Name

- Contents
  - Frameworks
  - Info.plist
  - MacOS
  - PkgInfo
  - Resources
  - Tools
    - about.txt
    - Documentation
    - Externals
    - License Agreement.txt
    - Open Source Licenses.txt
    - Plugins
      - Go_RevNet.rev
      - Navigator.rev
      - RegExBuilder 1.6.rev
      - revexample.rev
      - revMobGUI.livecode
      - revSmartProperties 1.2.rev
      - revSmartSave 1.1.rev
      - revTabRuler 1.2.rev
    - Release Notes.pdf
    - Resources
    - Runtime
    - Toolset
- LiveCode Setup.app
- Manifest

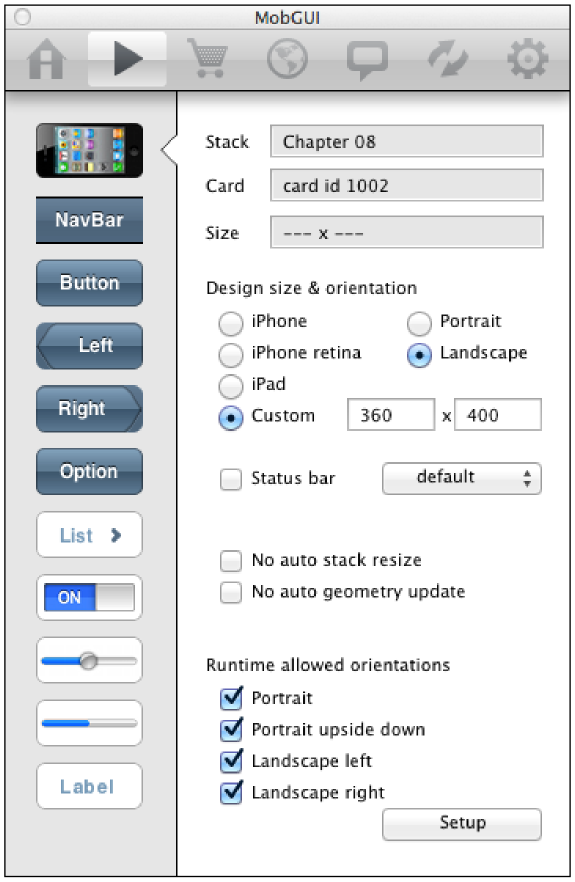
4. Open LiveCode and create a new main stack.
5. From the **Development** menu, navigate to **Plugins** | **revMobGUI**, as shown in the following screenshot:



6. The following screenshot shows you the main MobGUI interface that appears following the previous step of this recipe. On this interface, click on the second icon on the interface's banner. The second icon is a play button.



7. Next, you will see the interface illustrated in the following screenshot. You can use this interface to drag controls to cards in your LiveCode stack.



Stack: Chapter 08
Card: card id 1002
Size: --- x ---

Design size & orientation
iPhone
iPhone retina
iPad
[✓ Custom] 360 x 400
Portrait
[✓ Landscape]

Status bar: default

No auto stack resize
No auto geometry update

Runtime allowed orientations
[✓ Portrait]
[✓ Portrait upside down]
[✓ Landscape left]
[✓ Landscape right]

How it works...

In this recipe, we downloaded and installed the MobGUI plugin. We placed the plugin in the `Plugins` folder of our LiveCode application. This enables us to use the plugin for all LiveCode mobile apps.

There's more...

When you install new versions of LiveCode, you will need to manually copy and paste the `revMobGUI.livecode` file into the new version's `Plugins` folder.

See also

- The *Using a navigational bar* recipe
- The *Using a button* recipe
- The *Using a navigational button* recipe
- The *Using a slider* recipe
- The *Using a toggle button* recipe
- The *Using a list button* recipe
- The *Using a progress bar* recipe

Using a navigational bar

Navigational bars are typically rectangular bars that are located along the top of your app's user interface. It is common to have, as appropriate, **Back** and **Next** buttons on the left and right edges, respectively. In this recipe, you will learn how to create and use a navigational bar by using MobGUI.

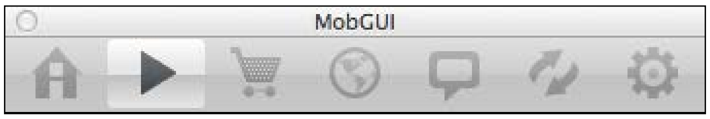
Getting ready

Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

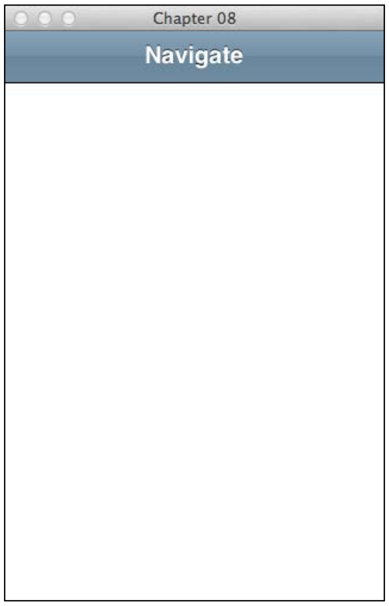
How to do it...

Follow the steps in this recipe to create a navigational bar for your mobile app.

1. Create a new main stack in LiveCode.
2. From the **Development** menu, select **Plugins** | **revMobGUI**.
3. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface.



4. Drag the NavBar icon from the MobGUI interface to your stack. This will result in a group being added to your stack.
5. Click on the new **NavBar** group so that it is selected.
6. In the MobGUI interface, change the name of the **NavBar** group to `Navigational Bar`.
7. In the MobGUI interface, change the label of the navigational bar to `Navigate`. Your stack should look like the following screenshot:

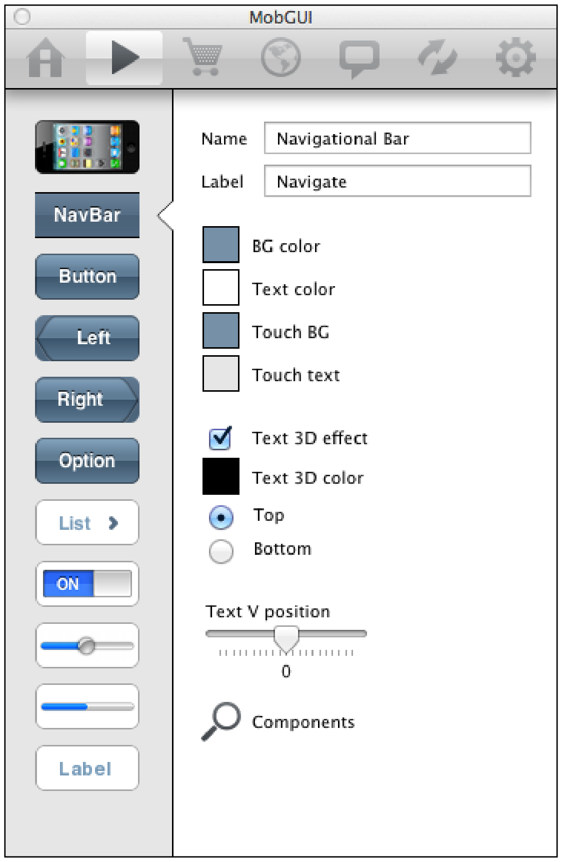


How it works...

The **Navigational Bar** group consists of a rectangle and a label. MobGUI uses a default color palette, and you have complete control of color properties. When you drag a **Navigational Bar** group to a card, it is autofitted to the top and center of that card.

There's more...

As illustrated in the following screenshot, there are additional settings that can be changed for your navigational bar. These include the background color, text color, color of the background when touched, color of the text when touched, 3D text effect, 3D text color, where to locate the bar (top or bottom), and where to position the text within the bar.



Name: Navigational Bar
Label: Navigate
BG color
Text color
Touch BG
Touch text
✓ Text 3D effect
Text 3D color
● Top
○ Bottom
Text V position: 0
Components

See also

- The *Setting up MobGUI* recipe

Using a button

In this recipe, you will learn how to use a button from the MobGUI plugin. MobGUI buttons are different from regular buttons. The buttons created using MobGUI are groups that contain graphics and a label.

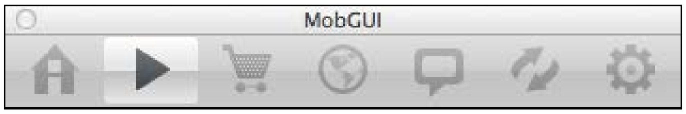
Getting ready

Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

How to do it...

Use the following steps to create a button for mobile apps using MobGUI:

1. Create a new main stack in LiveCode.
2. From the **Development** menu, navigate to **Plugins** | **revMobGUI**.
3. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface.



4. Drag the Button icon from the MobGUI interface to your stack. This will result in a group being added to your stack.
5. Right-click on the new **Button** group and select **Edit Script** from the pop-up menu. You will see the following script shell assigned to the **Button** group:

```
on touchEnd pId
  mobGUIUntouch the long id of me
  #visual effect push left very fast
  #go card "yourCard"
  #go prev card
  #go next card
end touchEnd


on touchRelease pId
  mobGUIUntouch the long id of me
end touchRelease


on touchStart pId
  mobGUITouch the long id of me
end touchStart


on mouseUp
  if the environment = "development" then touchEnd 1
end mouseUp


on mouseRelease
```

```
  if the environment = "development" then touchRelease 1
end mouseRelease


on mouseDown
  if the environment = "development" then touchStart 1
end mouseDown
```

How it works...

MobGUI provides button groups with some prescripting for use in your mobile apps. The scripts can easily be deleted or modified to suit the needs of your specific app. The scripts provided with the button object account for several user initiated actions: touch start, touch release, touch end, mouse down, mouse release, and mouse up.

See also

- The *Setting up MobGUI* recipe
- The *Using buttons for navigation* recipe in *Chapter 2*, *Human-computer Interfaces*

Using a navigational button

In this recipe, you will create two navigational buttons to switch between cards.

Getting ready

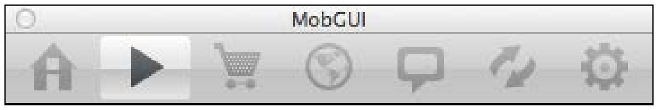
Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

How to do it...

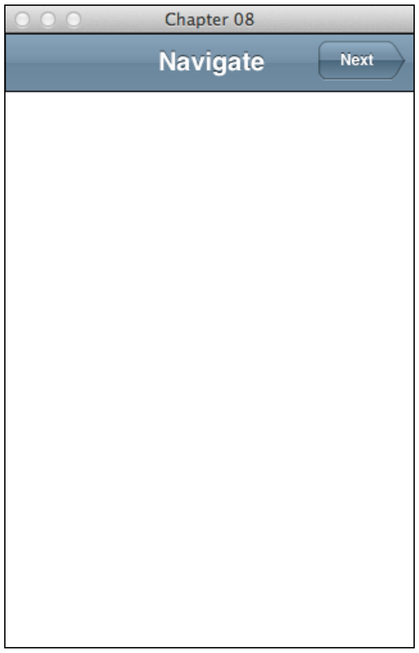
Use the steps in this recipe to create a navigational button that will be placed on a navigational bar of a mobile app:

1. Create a new main stack in LiveCode.
2. Change the name of the stack's card to `Home` using the property inspector.
3. From the **Development** menu, navigate to **Plugins** | **revMobGUI**.

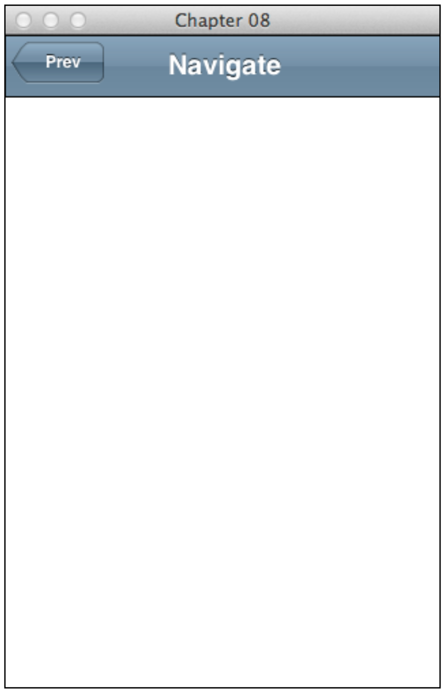
4. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface:



5. Drag the NavBar icon from the MobGUI interface to the **Home** card. This will result in a group being added to the card.
6. Click on the new **NavBar** group so that it is selected.
7. In the MobGUI interface, change the name of the **NavBar** group to `Navigational Bar`.
8. In the MobGUI interface, change the label of the navigational bar to `Navigate`.
9. Using the MobGUI interface, drag a **Right** button to the **Home** card and place it on the right-hand side section of the navigational bar.
10. Click on the new button so that it is selected.
11. Using the MobGUI interface, change the label of the **Right** button group to `Next`. Your interface should match the following screenshot:



12. Create a second card and name it `Final`.
13. Copy the navigational bar from the **Home** card and paste it on the **Final** card.
14. Using the MobGUI interface, drag a **Left** button to the **Final** card and place it on the left-hand side section of the navigational bar.
15. Click on the new button so that it is selected.
16. Using the MobGUI interface, change the label of the **Left** button group to `Prev`. Your interface should match the following screenshot:



17. Edit the script of the **Next** button on the **Home** card with the following code:

```
on mouseUp
  go to card "Final"
end mouseUp
```

18. Edit the script of the **Prev** button on the **Final** card with the following code:

```
on mouseUp
  go to card "Home"
end mouseUp
```

19. Test the app in a simulator or on an actual device.

How it works...

When a navigational button is added to a stack using MobGUI, the button is usually used to navigate between cards. The script shell provided by MobGUI for these buttons is listed as follows:

```
on touchEnd pId
  mobGUIUntouch the long id of me
  #visual effect push right very fast
  #go card "yourCard"
  #go prev card
end touchEnd


on touchRelease pId
  mobGUIUntouch the long id of me
end touchRelease


on touchStart pId
  mobGUITouch the long id of me
end touchStart


on mouseUp
  if the environment = "development" then touchEnd 1
end mouseUp


on mouseRelease
  if the environment = "development" then touchRelease 1
end mouseRelease


on mouseDown
  if the environment = "development" then touchStart 1
end mouseDown
```

See also

- The *Setting up MobGUI* recipe
- The *Using buttons for navigation* recipe in *Chapter 2*, *Human-computer Interfaces*

Using a slider

In this recipe, you will learn how to use a MobGUI slider. Sliders can be used to show a range of values.

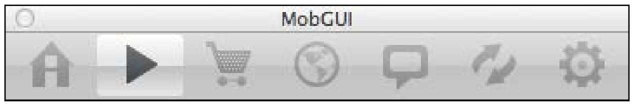
Getting ready

Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create a slider that can be used to select a numeric value:

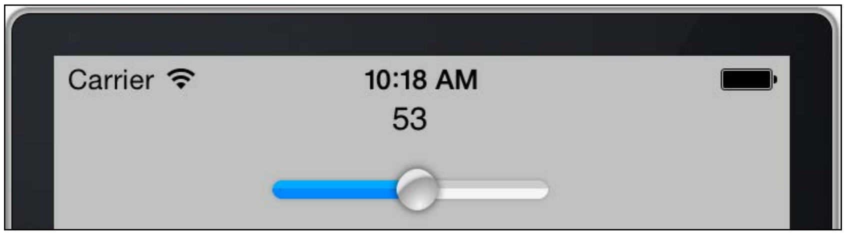
1. Create a new main stack in LiveCode.
2. From the **Development** menu, navigate to **Plugins** | **revMobGUI**.
3. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface:



4. Drag the slider icon from the MobGUI interface to the card. This will result in a group being added to the card.
5. Using the property inspector, position the slider so that it is at location `155,58`.
6. Click on the new slider group so that it is selected.
7. In the MobGUI interface, change the name of the slider to `mySlider`.
8. Drag a standard label from the LiveCode Tools palette and set the following properties:
   - **Location**: `155,24`
   - Align text center
   - **Name**: `myLabel`
9. Edit the slider's script to match the following:

```
on sliderDrag pValue
  set the text of fld "myLabel" to pValue
on sliderDrag pValue
  set the text of fld "myLabel" to pValue
end sliderDrag   end sliderDrag
```

10. Run the app in a simulator and experiment with the slider's position. The label should be updated every time the slider is moved, as shown in the following screenshot:



How it works...

The MobGUI slider is a group of three graphics: `letfBar`, `rightBar`, and a button that makes up a mobile-ready slider. Like other MobGUI controls, the slider comes with script shells to make programming the object easier.

See also

- The *Setting up MobGUI* recipe

Using a toggle button

In this recipe, you will learn how to use a MobGUI toggle button, which is also referred to as a switch. You will create a mood toggle to display how you are feeling.

Getting ready

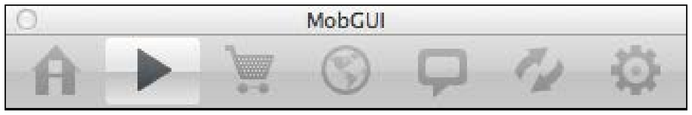
Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

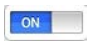
How to do it...

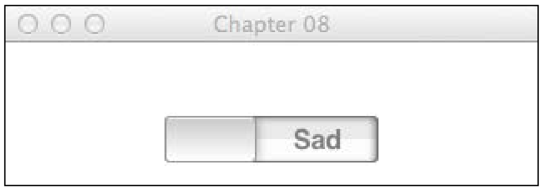
Follow the steps in this recipe to create a toggle button and allow the user to select between mood values of **Happy** and **Sad**:

1. Create a new main stack in LiveCode.
2. From the **Development** menu, navigate to **Plugins** | **revMobGUI**.

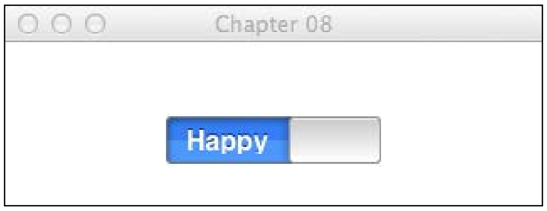
3. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface:



4. Drag the  icon from the MobGUI interface to the card. This will result in a group being added to the card.
5. Using the property inspector, change the width to `128`.
6. Using the property inspector, change the position of the group so that it is at location `160,58`.
7. Click on the new toggle group so that it is selected.
8. In the MobGUI interface, change the name of the toggle to `myMood`.
9. In the MobGUI interface, change the labels to `Happy` and `Sad`.
10. Run the app in your IDE, a simulator, or an actual device to test your new toggle. You should be able to switch between the two moods as illustrated in the following two screenshots, with the first screenshot showing you the **Sad** mood:



The following screenshot shows you the **Happy** mood:



How it works...

The MobGUI toggle/switch is a group of four objects: right group, left group, button group, and a border graphic. These objects make up the mobile-ready toggle. MobGUI permits us to change the name, labels, colors, and more, giving us great design and programmatic control.

See also

- The *Setting up MobGUI* recipe

Using a list button

In this recipe, you will learn how to use a MobGUI list button. This button can be used as a mobile interface object. For this recipe, you will create a list button that allows the user to select a shipping option.

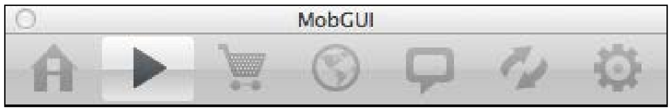
Getting ready

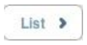
Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

How to do it...

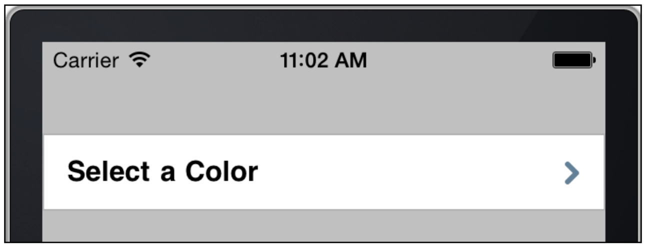
Follow the steps in this recipe to create a list button for a mobile app:

1. Create a new main stack in LiveCode.
2. From the **Development** menu, select **Plugins** | **revMobGUI**.
3. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface.



4. Drag the  icon from the MobGUI interface to the card. This will result in a group being added to the card.
5. Using the property inspector, change the width to `320`.
6. Using the property inspector, change the position of the group so that it is at location `160,74`.
7. Click on the new list button group so that it is selected.
8. In the MobGUI interface, change the name of the list button to `myButton`.
9. In the MobGUI interface, change the label to `Select a Color`.
10. In the MobGUI interface, change the text color to black.

11. Run the app in your IDE, a simulator, or an actual device (see the following screenshot):



How it works...

When a list button is added to a stack using MobGUI, the button is usually used to navigate to another card where options related to the list button's label are presented to the user. The script shell provided by MobGUI for these buttons is listed as follows:

```
on touchEnd pId
  mobGUIUntouch the long id of me
  #visual effect push left very fast
  #go card "yourCard"
  #go next card
end touchEnd


on touchRelease pId
  mobGUIUntouch the long id of me
end touchRelease


on touchStart pId
  mobGUITouch the long id of me
end touchStart


on mouseUp
  if the environment = "development" then touchEnd 1
end mouseUp


on mouseRelease
  if the environment = "development" then touchRelease 1
end mouseRelease


on mouseDown
  if the environment = "development" then touchStart 1
end mouseDown
```

There's more...

You can use the `on touchEnd` command to indicate which card you want the app to display when the user swipes or touches the list button.

See also

- The *Setting up MobGUI* recipe

Using a progress bar

Progress bars are great for presenting the user with a visual status such as load time. MobGUI provides a nice slider that is mobile ready. In this recipe, you will learn how to use a MobGUI progress bar.

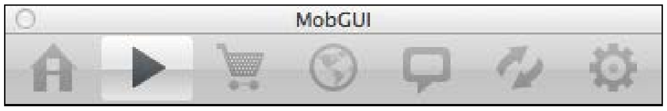
Getting ready

Before using this recipe, you will need to have the MobGUI plugin downloaded and installed on your development computer. See the *Setting up MobGUI* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create a progress bar using MobGUI:

1. Create a new main stack in LiveCode.
2. From the **Development** menu, navigate to **Plugins** | **revMobGUI**.
3. Click on the play button, shown in the following screenshot, from the icons on the MobGUI interface:



4. Drag the icon from the MobGUI interface to the card. This will result in a group being added to the card.
5. Click on the new slider group so that it is selected.
6. In the MobGUI interface, change the name of the slider to `myProgressBar`.
7. You can change the value of the progress bar with the following line of code:

   ```
   set the uValue of grp "myProgressBar" to 75
   ```

How it works...

The progress bar in MobGUI is a group of several objects. This group has custom properties that you can set and change in your mobile apps. These custom properties include `uValue`, which you used in this recipe.

See also

- The *Setting up MobGUI* recipe

9

Using Animation Engine

In this chapter, we will cover the following recipes:

- Setting up Animation Engine
- Moving objects along a line
- Moving objects along a polygonal path
- Moving objects along an elliptical path
- Moving objects along a circular path
- Stopping a moving object
- Calculating the distance between two points
- Using speed
- Using collision listeners
- Simulating gravity

Introduction

In this chapter, you will learn how to harness the power of Animation Engine 5, which is a LiveCode extension, to animate objects and employ the game and simulation functionality in your mobile apps. The Animation Engine 5 extension is a commercial product that does not come with a commercial or community LiveCode license. The extension can be purchased from the LiveCode web store.

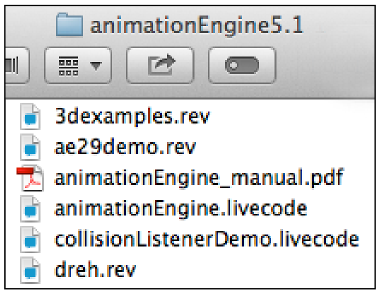
Setting up Animation Engine

This recipe walks you through downloading, installing, and setting up Animation Engine so that you can use it within your LiveCode mobile applications.

How to do it...

Follow the steps in this recipe so that Animation Engine is available for you within LiveCode as you develop your mobile applications:

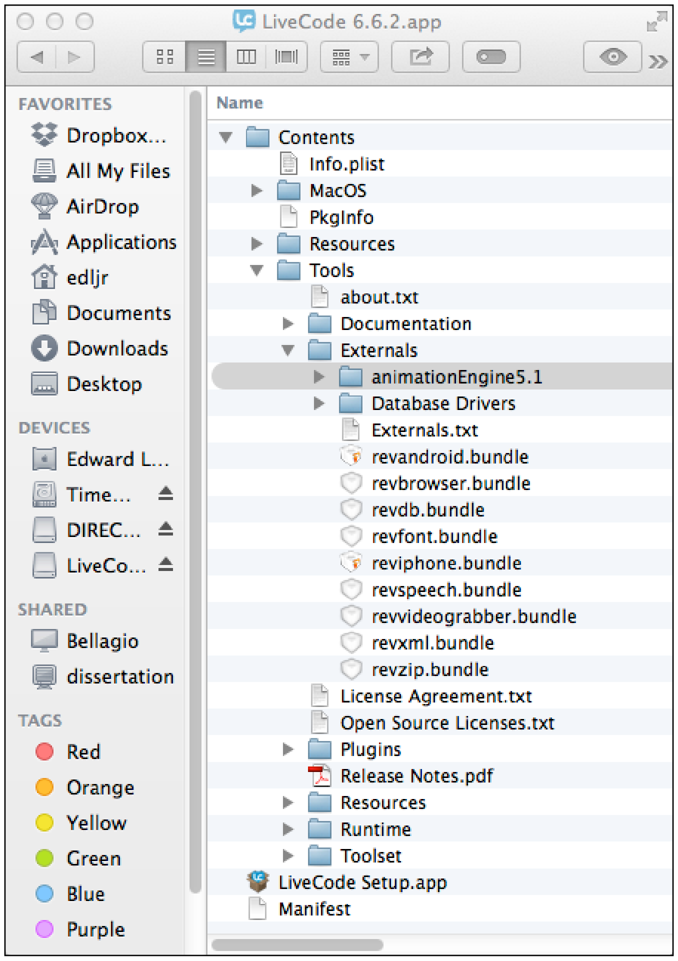
1. Download the Animation Engine extension from the LiveCode store.
   You should now have the `animationEngine5.1.zip` compressed file on your development computer.
2. Open the `animationEngine5.1.zip` compressed file so that the file contents are revealed (see the following screenshot):



animationEngine5.1

3dexamples.rev
ae29demo.rev
animationEngine_manual.pdf
animationEngine.livecode
collisionListenerDemo.livecode
dreh.rev

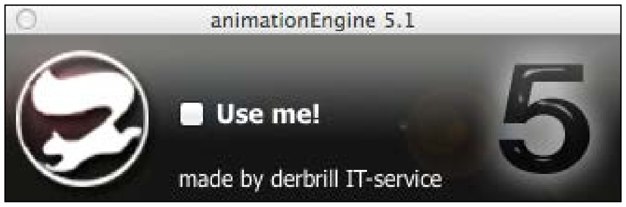
3. Open the LiveCode application package to show contents.
4. Move the `animationEngine.livecode` stack to the LiveCode `Externals` folder/directory, as shown in the following screenshot:



```
Contents
    Info.plist
    MacOS
    PkgInfo
    Resources
    Tools
        about.txt
        Documentation
        Externals
            animationEngine5.1
            Database Drivers
            Externals.txt
            revandroid.bundle
            revbrowser.bundle
            revdb.bundle
            revfont.bundle
            reviphone.bundle
            revspeech.bundle
            revvideograbber.bundle
            revxml.bundle
            revzip.bundle
        License Agreement.txt
        Open Source Licenses.txt
        Plugins
        Release Notes.pdf
        Resources
        Runtime
        Toolset
LiveCode Setup.app
Manifest
```

5. Open LiveCode.
6. Create a new main stack.
7. Next, open the `animationEngine` stack you previously downloaded to your development computer. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file in the LiveCode `Externals` folder/directory.

8. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



How it works...

We downloaded the Animation Engine stack and placed it in the required directory/file folder so that LiveCode knows where to find it. When we open the Animation Engine external when running LiveCode, it works alongside your mobile app. Once you open the `animationEngine` stack, you will have full access to its functionalities.

Animation Engine 5 requires LiveCode Version 5.0 or higher.

See also

- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Moving objects along a line

In this recipe, you will learn how to animate an object by moving it along a straight line. To accomplish our goal, we will create an oval and two buttons. The first button will animate the oval, and at its end state, change its background color. The second button will reset the oval to its original position and color.

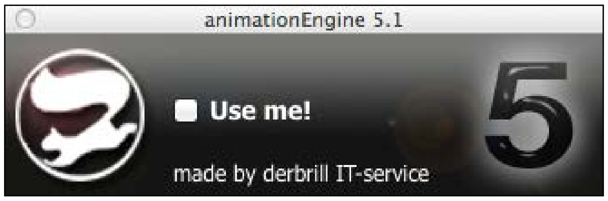
Getting ready

Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create and animate a graphic:

1. Open LiveCode.
2. Create a new main stack.
3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myButton1`
    - **Label**: `Animate`
    - **Location**: `106,29`
6. Drag a second button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myButton2`
    - **Label**: `Reset`
    - **Location**: `254,29`

7. Drag an oval graphic to the stack's card, and set the following preferences:
   - **Name**: `myOval`
   - **Opaque**: Keep this checked
   - **Height**: `34`
   - **Width**: `34`
   - **Location**: `35, 85`
   - backgroundColor: Green
8. Add the following code to the `myButton1` button (**Animate**):

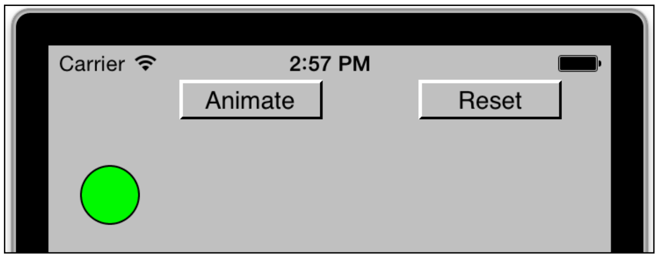
```
on mouseUp
  aeMoveTo the name of graphic "myOval",300,85,1000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

9. Add the following code to the `myButton2` button (**Reset**):

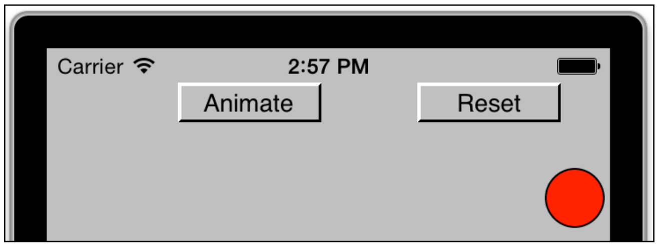
```
on mouseUp
  set the loc of grc "myOval" to 35,85
  set the backgroundColor of grc "myOval" to green
end mouseUp
```

10. Test your mobile app in a simulator or on an actual device. The next two screenshots illustrate the beginning and ending animation states.

    The following screenshot illustrates the beginning animation state:



The following screenshot illustrates the ending animation state:



How it works...

We used the `aeMoveTo` handler to animate our oval along an imaginary line. The `aeMoveTo` handler has the following syntax:

```
aeMoveTo controlName, x,y, duration, [easingEffect]
```

The handler's parameters consist of the following:

- `controlName`: This is a reference to a control or stack (that is, the name of the `myOval` graphic)
- `x,y`: This is the parameter for the (x, y) coordinates that the control or stack is to be moved to
- `duration`: This tells you how long, in milliseconds, the move should take from start to finish
- `easingEffect`: This optional parameter can be `in`, `out`, `inOut`, `bounce`, or `overshoot`

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Moving objects along a polygonal path

In this recipe, you will create a graphic and write a LiveCode script, using Animation Engine, to move the graphic along a polygonal path.

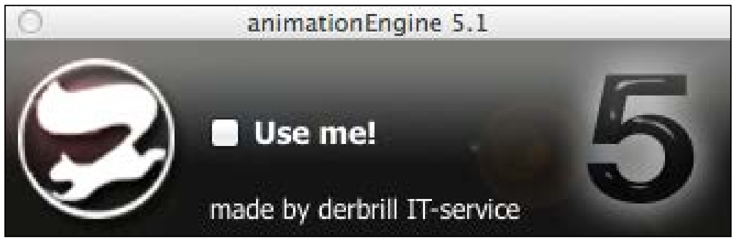
Getting ready

Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create a graphic and, using the LiveCode script and Animation Engine, move it along a polygonal path:

1. Open LiveCode.
2. Create a new main stack.
3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the preceding steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myMPButton`
    - **Label**: `Move Polygonal`
    - **Width**: `126`
    - **Location**: `77, 41`
6. Drag an oval graphic to the stack's card, and set the following preferences:
    - **Name**: `myDot`
    - **Opaque**: Keep this checked
    - **Height**: `15`
    - **Width**: `15`
    - **Location**: `67, 111`
    - backgroundColor: Red
7. Add the following code to the `myMPButton` button:

```
on mouseUp
  if the flag of me is empty then set the flag of me
    to false
  set the flag of me to not the flag of me
  moveDot
end mouseUp

on moveDot
  if the movePolygonal["moveDone"] of grc "myDot" is
    true then
      set the loc of grc "myDot" to the
        movePolygonal["endpoint"] of grc "myDot"
      set the movePolygonal["moveDone"] of grc "myDot" to
        false
  else
    if the flag of me then
      send "movePolygonal" to grc "myDot"
      send "moveDot" to me in 5 milliseconds
    end if
  end if
end moveDot
```

8. Test the code in the simulator or on an actual device. You'll see that when the button is depressed, the `myDot` graphic moves along a polygonal path.

How it works...

The `movePolygonal` handler has a set of seven custom properties that you refer to using the array notation as we did in the code provided in step 7 of the previous section. The custom properties are as follows:

- `current`: This is the current point along the polygonal path
- `endpoint`: This is the end point (position) of the current line
- `isDistance`: This tells you how many pixels the object has moved along the current line
- `moveDone`: This will be `true` or `false` depending upon whether the object completed the move
- `pointList`: This is each (x,y) point of the line
- `startPoint`: This is the (x,y) point of the start of the line
- `step`: This gives you the direction and speed of the object that moves along the path

The button labeled **Move Polygonal** serves as a toggle to start and stop the animation.

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Moving objects along an elliptical path

In this recipe, you will create a graphic and write a LiveCode script, using Animation Engine, to move the graphic along an elliptical path.

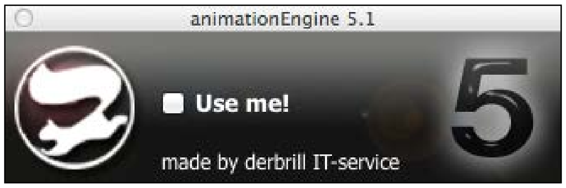
Getting ready

Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create a graphic and, using the LiveCode script and Animation Engine, move it along an elliptical path:

1. Open LiveCode.
2. Create a new main stack.
3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myMEButton`
    - **Label**: `Move Elliptical`
    - **Width**: `126`
    - **Location**: `77`, `41`
6. Drag an oval graphic to the stack's card, and set the following preferences:
    - **Name**: `myDot`
    - **Opaque**: Keep this checked
    - **Height**: `15`
    - **Width**: `15`
    - **Location**: `67`, `111`
    - backgroundColor: Red
7. Add the following code to the `myMEButton` button:

```
on mouseUp
  if the flag of me is empty then set the flag of me
    to false
  set the flag of me to not the flag of me
  moveDot
end mouseUp

on moveDot
  send "moveElliptical" to grc "myDot"
  if the flag of me then
    send "moveDot" to me in 5 milliseconds
  end if
end moveDot
```

8. Test the code in the simulator or on an actual device. You'll see that when the button is depressed, the `myDot` graphic moves along an elliptical path.

How it works...

The `moveElliptical` handler has a set of six custom properties that you refer to using the array notation as we did in the code provided in step 7 of the previous section. The custom properties are as follows:

- `centerX`: This is the specific (x) point of the center point
- `centerY`: This is the specific (y) point of the center point

- `isAngle`: This is the center angle degree of the ellipse
- `radiusX`: This is the ellipse's x radius
- `radiusY`: This is the ellipse's y radius
- `step`: This is the direction and speed of the object that moves along the path

[The button labeled **Move Elliptical** serves as a toggle to start and stop the animation.]

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Moving objects along a circular path

In this recipe, you will create a graphic and write a LiveCode script, using Animation Engine, to move the graphic along a circular path.

Getting ready

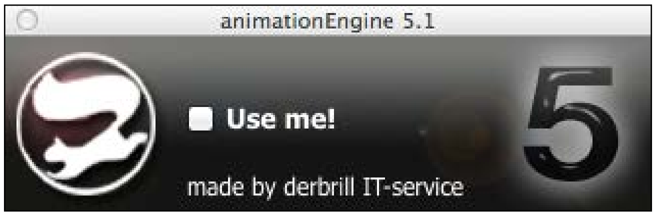
Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create a graphic and, using the LiveCode script and Animation Engine, move it along a circular path:

1. Open LiveCode.
2. Create a new main stack.

3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `myMCButton`
   - **Label**: `Move Elliptical`
   - **Width**: `126`
   - **Location**: `77`, `41`
6. Drag an oval graphic to the stack's card, and set the following preferences:
   - **Name**: `myDot`
   - **Opaque**: Keep this checked
   - **Height**: `15`
   - **Width**: `15`
   - **Location**: `67`, `111`
   - backgroundColor: Red
7. Add the following code to the `myMCButton` button:

```
on mouseUp
  if the flag of me is empty then set the flag of me to
    false
  set the flag of me to not the flag of me
  moveDot
end mouseUp

on moveDot
  send "moveCircular" to grc "myDot"
  if the flag of me then
    send "moveDot" to me in 5 milliseconds
  end if
end moveDot
```

8. Test the code in the simulator or on an actual device. You'll see that when the button is pressed, the `myDot` graphic moves along an elliptical path.

How it works...

The `moveCircular` handler has a set of five custom properties that you refer to using the array notation, as we did in the code provided in step 7 of the previous section. The custom properties are as follows:

- `centerX`: This is the specific (x) point of the center point
- `centerY`: This is the specific (y) point of the center point
- `isAngle`: This is the center angle degree of the ellipse circle
- `isRadius`: This is the radius of the circle
- `step`: This is the direction and speed of the object that moves along the path

The button labeled **Move Circular** serves as a toggle to start and stop the animation.

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Stopping a moving object

In this recipe, we will create a graphic object and three buttons. The first button will animate the graphic object, the second will stop it, and the third will reset the graphical object to its original position.

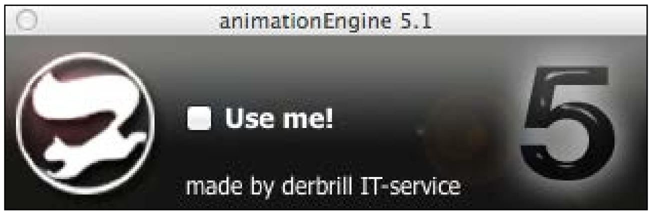
Getting ready

Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create, animate, and stop an object:

1. Open LiveCode.
2. Create a new main stack.
3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myStartButton`
    - **Label**: `StartMoving`
    - **Width**: `126`
    - **Location**: `73, 25`
6. Drag a second button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myStopButton`
    - **Label**: `Stop Moving`
    - **Location**: `203, 25`

7. Drag a third button to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `myResetButton`
   - **Label**: `Reset`
   - **Location**: `313,25`
8. Drag an oval graphic to the stack's card, and set the following preferences:
   - **Name**: `myOval`
   - **Opaque**: Keep this checked
   - **Height**: `34`
   - **Width**: `34`
   - **Location**: `35,85`
   - backgroundColor: Green
9. Add the following code to the `myStartButton` button (**Start Moving**):

```
on mouseUp
  aeMoveTo the name of graphic "myOval",300,85,5000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

10. Add the following code to the `myStopButton` button (**Stop Moving**):

```
on mouseUp
  aeStopMoving the name of grc "myOval"
end mouseUp
```

11. Add the following code to the `myResetButton` button (**Reset**):

```
on mouseUp
  set the loc of grc "myOval" to 35,85
  set the backgroundColor of grc "myOval" to green
end mouseUp
```

12. Test your mobile app in a simulator or on an actual device.

How it works...

The `aeStopMoving` handler can stop an object that is currently being moved by the `aeMoveTo` handler. In this recipe, we used the `aeStopMoving` handler to immediately stop the movement of `grc "myOval"`.

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Calculating the distance between two points

In this recipe, you will create two graphics using LiveCode and Animation Engine. You will also create a button that animates the graphics. Finally, you will create a button that calculates the distance between the two graphics. Being able to calculate the distance between two objects can come in handy for game and simulation applications.

Getting ready

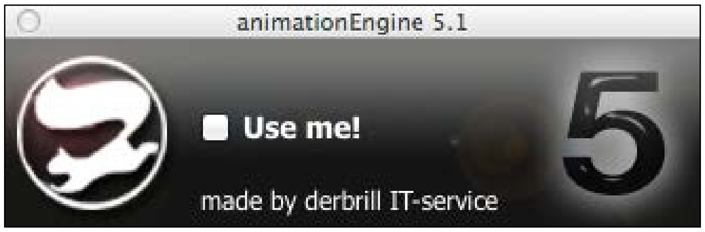
Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to calculate the distance between two points:

1. Open LiveCode.
2. Create a new main stack.
3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.

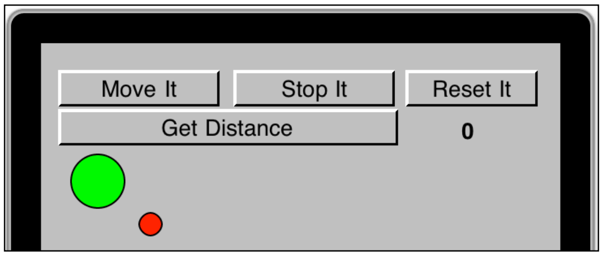
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myMoveItButton`
    - **Label**: `Move It`
    - **Width**: `100`
    - **Location**: `60,27`
6. Drag a second button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myStopItButton`
    - **Label**: `Stop It`
    - **Width**: `100`
    - **Location**: `168,27`
7. Drag a third button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myResetItButton`
    - **Label**: `Reset It`
    - **Width**: `82`
    - **Location**: `265,27`
8. Drag a fourth button to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myCalculateButton`
    - **Label**: `Get Distance`
    - **Width**: `210`
    - **Location**: `115,51`

9. Drag a label field to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `myDistanceLabel`
   - **Width**: `78`
   - **Location**: `263, 51`
   - **Contents**: `0`
   - Align text center
   - Set the text style as bold
10. Drag an oval graphic to the stack's card, and set the following preferences:
    - **Name**: `myOval`
    - **Opaque**: Keep this checked
    - **Height**: `34`
    - **Width**: `34`
    - **Location**: `35, 85`
    - backgroundColor: Green
11. Drag an oval graphic to the stack's card, and set the following preferences:
    - **Name**: `myDot`
    - **Opaque**: Keep this checked
    - **Height**: `15`
    - **Width**: `15`
    - **Location**: `67, 111`
    - backgroundColor: Red

    Your interface should resemble the following screenshot:



12. Add the following code to the `myMoveItButton` button (**Move It**):

```
on mouseUp
  aeMoveTo the name of graphic "myOval",300,85,5000
  set the backgroundColor of grc "myOval" to red
  --
  if the flag of me is empty then set the flag of me to
    false
  set the flag of me to not the flag of me
  moveDot
end mouseUp

on moveDot
  send "moveCircular" to grc "myDot"
  if the flag of me then
    send "moveDot" to me in 5 milliseconds
  end if
end moveDot
```

13. Add the following code to the `myStopItButton` button (**Stop It**):

```
on mouseUp
  aeMoveTo the name of graphic "myOval",300,85,5000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

14. Add the following code to the `myResetItButton` button (**Reset It**):

```
on mouseUp
  set the loc of grc "myOval" to 35,85
  set the backgroundColor of grc "myOval" to green
  --
  set the loc of grc "myDot" to 67,111
end mouseUp
```

15. Add the following code to the `myCalculateButton` button (**Get Distance**):

```
on mouseUp
  get distance (the loc of grc "myOval", the loc of grc
    "myDot")
  put it into fld "myDistanceLabel"
end mouseUp
```

16. Test your mobile app in a simulator or on an actual device. The `myDistanceLabel` field will be updated each time the `myCalculateButton` button is depressed.

How it works...

The `get distance` function calculates the distance between two objects. This function can be used to automatically update an onscreen display, or it can be used in other calculations such as when an enemy's weapon is within range (gaming reference).

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Using speed

In this recipe, you will animate a graphic by alternating between three buttons. Each button will animate the graphic in a linear motion but each at a different frame rate. The default frame rate for all animations is 25 frames per second (FPS). Depending upon your animation needs, you might need to decrease or increase the speed of selected animations in your mobile apps.

Getting ready

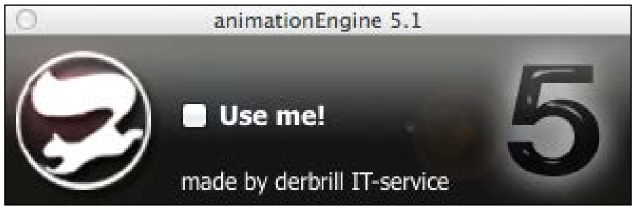
Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to change the speed of animations using Animation Engine:

1. Open LiveCode.
2. Create a new main stack.

3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag a button to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `btn10`
   - **Label**: `10 FPS`
   - **Width**: `80`
   - **Location**: `48,21`
6. Drag a second button to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `btn25`
   - **Label**: `25 FPS`
   - **Width**: `80`
   - **Location**: `48,47`
7. Drag a third button to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `btn50`
   - **Label**: `50 FPS`
   - **Width**: `80`
   - **Location**: `48,73`
8. Drag a fourth button to the stack's card, and set the following preferences using the property inspector:
   - **Name**: `myReset`
   - **Label**: `Reset`
   - **Width**: `80`
   - **Location**: `48,99`

9. Drag an oval graphic to the stack's card, and set the following preferences:
    - **Name**: `myOval`
    - **Opaque**: keep it checked
    - **Height**: `34`
    - **Width**: `34`
    - **Location**: `35, 85`
    - backgroundColor: Green
10. Add the following code to the `btn10` button (**10 FPS**):

```
on mouseUp
  aeSetFrameRate 10
  aeMoveTo the name of graphic "myOval",300,133,5000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

11. Add the following code to the `btn25` button (**25 FPS**):

```
on mouseUp
  aeSetFrameRate 25
  aeMoveTo the name of graphic "myOval",300,133,5000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

12. Add the following code to the `btn50` button (**50 FPS**):

```
on mouseUp
  aeSetFrameRate 50
  aeMoveTo the name of graphic "myOval",300,133,5000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

13. Add the following code to the `myReset` button (**Reset**):

```
on mouseUp
  aeSetFrameRate 10
  aeMoveTo the name of graphic "myOval",300,133,5000
  set the backgroundColor of grc "myOval" to red
end mouseUp
```

14. Test the code in the simulator or on an actual device. You'll notice that the speed of the animation decreases as the FPS setting is increased.

How it works...

The `aeSetFrameRate` handler only works with objects that are being moved with the `aeMoveTo` handler. In this recipe, we used the `aeSetFrameRate` handler to change the FPS setting, which impacted the speed of the animation.

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using collision listeners* recipe
- The *Simulating gravity* recipe

Using collision listeners

A collision listener is a type of physics software that detects the collisions between two objects. In this recipe, you will create two graphics and listen for collisions between them. Detecting collisions (we do this by "listening" for them) in a mobile app can be very useful when developing games and simulations. As a game example, your hero shoots an arrow at an enemy; we want to know whether the arrow collided with the enemy's armor or body.

Getting ready

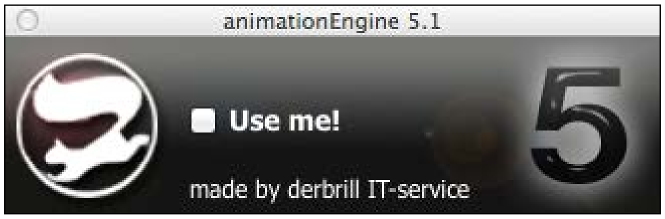
Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe discussed earlier in this chapter.

How to do it...

Follow the steps in this recipe to create two objects and add the necessary script to detect collisions between them:

1. Open LiveCode.
2. Create a new main stack.

3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



5. Drag an oval graphic to the stack's card, and set the following preferences:
   - **Name**: `gr1`
   - **Opaque**: Keep this checked
   - **Height**: `50`
   - **Width**: `50`
   - **Location**: `183,155`
   - backgroundColor: Red
6. Drag a second oval graphic to the stack's card, and set the following preferences:
   - **Name**: `gr2`
   - **Opaque**: Keep this checked
   - **Height**: `34`
   - **Width**: `34`
   - **Location**: `241,155`
   - backgroundColor: Red
7. Drag a button to the stack's card, and set the following preferences:
   - **Name**: `collisionButton`
   - **Label**: `Start`
   - **Width**: `94`
   - **Location**: `51,19`

8. Drag a label field to the stack's card, and set the following preferences using the property inspector:
    - **Name**: `myOutputLabel`
    - **Width**: `230`
    - **Location**: `121,48`
    - **Contents**: `<blank>`
    - Align text left
    - Set the text style as bold
9. Add the following code to the card:

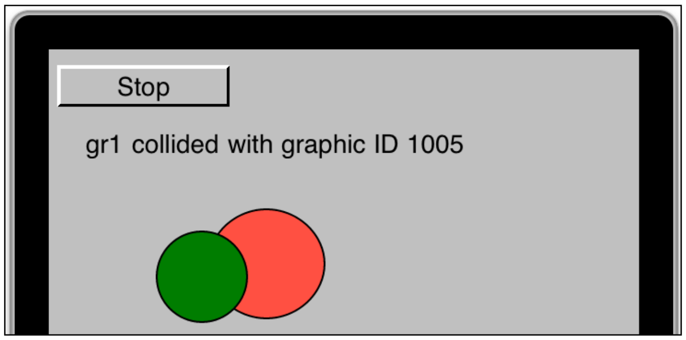
```
on aeCollision pObjects
  put the short name of the target && "collided with
    graphic ID " & word 3 of pObjects into fld
    "myOutputLabel"
end aeCollision

on constrainRectangularCallBack
  local tmpObjects
  put aeCollidingObjects() into tmpObjects
    if the keys of tmpObjects is empty then
      put empty into fld "myOutputLabel"
    end if
end constrainRectangularCallBack
```

10. Add the following code to the `collisionButton` button:

```
on mouseUp pMouseBtnNo
  aeStopListeningForCollisions
  local grcList
  set the flag of me to not the flag of me
  if the flag of me then
    set the constrainRectangular of grc "gr1" to the rect
      of this card
    set the constrainRectangular of grc "gr2" to the rect
      of this card
    --
    put the long id of graphic 2 after grcList
    set the aeListenForCollisionsWith of graphic 1 to
      grcList
    aeStartListeningForCollisions
    set the label of me to "Stop"
  else
```

```
        aeStopListeningForCollisions
        set the label of me to "Start"
      end if
    end mouseUp
```

11. Test your application in the simulator or on an actual device. You will note that the app listens for collisions as you move one or both graphic objects, as shown in the following screenshot:



gr1 collided with graphic ID 1005

How it works...

For this recipe, we created two graphics and ensured that their opaque value was set to true. We also constrained their movement to the confines of the card. These steps ensured that the graphics were draggable. Next, we created a list (`grcList`) of objects for our listener to check collisions with.

There's more...

When you download a copy of Animation Engine, you receive full documentation as well as sample stacks to include a Collision Listener Demo stack.

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe

- The *Using Speed* recipe
- The *Simulating gravity* recipe

Simulating gravity

Animation Engine makes simulating gravity in mobile apps an easy task. In this recipe, we will create an orange circular graphic and a button. The button will contain a LiveCode script that will animate the graphic to simulate a bouncing ball.

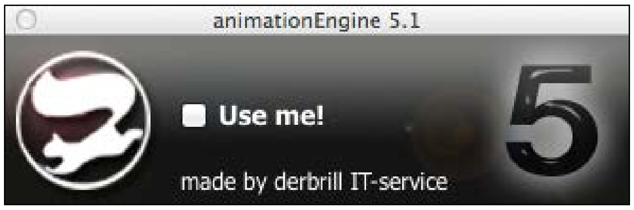
Getting ready

Before using this recipe, you will need to have the Animation Engine external downloaded and available on your development computer. See the *Setting up Animation Engine* recipe earlier in this chapter.

How to do it...

Follow the steps in this recipe to create objects and a script to simulate gravity in a mobile application:

1. Open LiveCode.
2. Create a new main stack.
3. Open the `animationEngine` stack. With LiveCode already running, you can simply double-click on the `animationEngine.livecode` file.
4. Once you complete the previous steps, you will see the animationEngine window with a **Use me!** checkbox (see the following screenshot). Check that box.



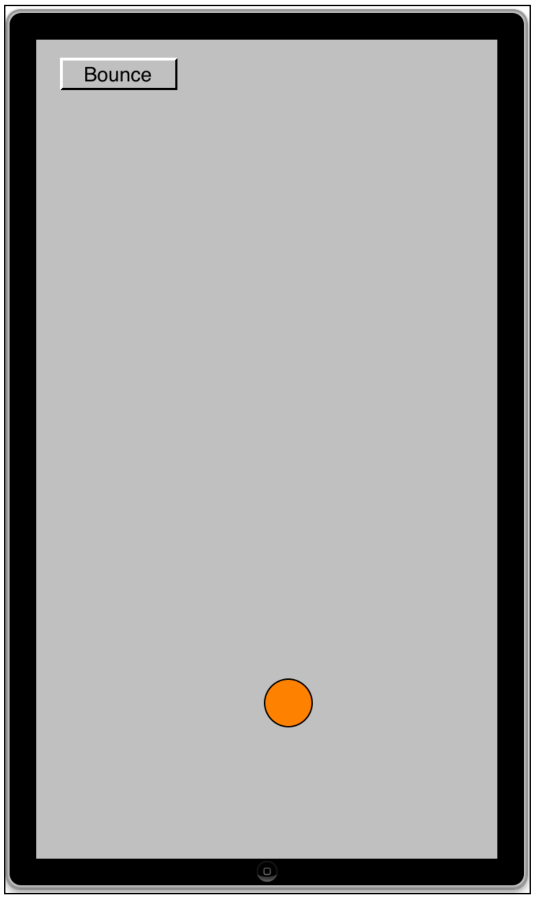
5. Drag an oval graphic to the stack's card, and set the following preferences:
   - **Name**: `myBall`
   - **Opaque**: Keep this checked
   - **Height**: `50`
   - **Width**: `50`
   - **Location**: `175,27`
   - backgroundColor: Orange

6. Drag a button to the stack's card, and set the following preferences:
   - **Name**: `myButton`
   - **Label**: `Bounce`
   - **Width**: `82`
   - **Location**: `57,23`

7. Add the following code to the `myButton` button:

```
on mouseUp
  set the loc of grc "myBall" to 175,27
  aeMoveTo the name of grc "myBall", 175,460, 2000, "bounce"
end mouseUp
```

8. Test the app in a simulator or on an actual device. When you press the **Bounce** button, the ball will start at the top of the screen, fall to an imaginary line, and bounce until it comes to a stop (see the following screenshot):



How it works...

We simulated gravity with a bouncing ball. We did this with two lines of code: one native to LiveCode and one specific to Animation Engine. The first line of code resets the location of the graphic to the top of the screen. The second line of code uses the `aeMoveTo` handler with the optional easing effect parameter of `bounce`.

See also

- The *Setting up Animation Engine* recipe
- The *Moving objects along a line* recipe
- The *Moving objects along a polygonal path* recipe
- The *Moving objects along an elliptical path* recipe
- The *Moving objects along a circular path* recipe
- The *Stopping a moving object* recipe
- The *Calculating the distance between two points* recipe
- The *Using speed* recipe
- The *Using collision listeners* recipe

10 Miscellaneous

In this chapter, we will cover the following recipes:

- Adding numbers
- Subtracting numbers
- Multiplying numbers
- Dividing numbers
- Using advanced math
- Randomizing numbers
- Opening a web page
- Querying web data
- Using the geometry manager
- Using invisible objects
- Taking snapshots of a card
- Taking snapshots of an area on a card
- Detecting the operating system

Introduction

In this chapter, you have access to several mathematics-related recipes to include simple math operations, advanced math, and random number generation. In addition, there are several other miscellaneous recipes that did not concisely fit into other chapters. These include the *Opening a web page* recipe, the *Querying web data* recipe, the *Using invisible objects* recipe, the *Using the geometry manager* recipe, and more. Also, in this chapter, you will learn how to take screen snapshots.

Adding numbers

Adding numbers using LiveCode is a simple task. To demonstrate how to add two numbers, we will create a user interface that accepts two numbers and, when the equals sign is selected, the sum of the two numbers will be displayed.

How to do it...

Follow the steps in this recipe to gain experience in adding two numbers with a mobile app written in LiveCode:

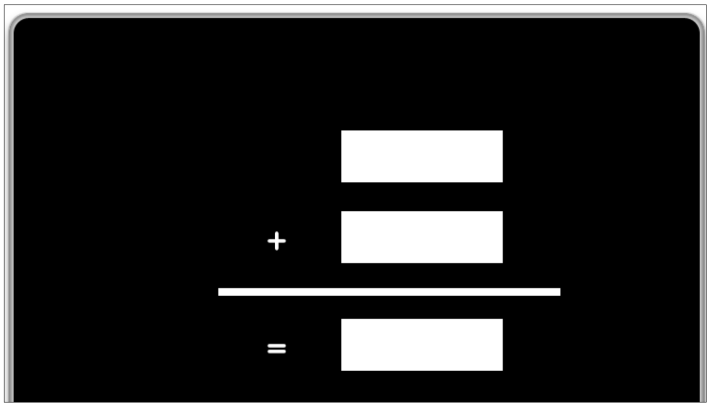
1. Open LiveCode and create a new main stack.
2. Set the background color of the default card to black.
3. Drag a new text entry field to the card and set the following properties:
   - **Name**: `fld_nbr1`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `194, 53`
   - **Font**: **Courier**
   - Text size: **18**
   - Align text right
4. Drag a second text entry field to the card and set the following properties:
   - **Name**: `fld_nbr2`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `194, 95`
   - **Font**: **Courier**
   - Text size: **18**
   - Align text right
5. Drag a third text entry field to the card and set the following properties:
   - **Name**: `fld_nbr3`
   - **Width**: `88`
   - **Height**: `31`

- **Location**: `194, 151`
- **Font**: **Courier**
- Text size: **18**
- Align text right

6. Drag a new label field to the card and set the following properties:
    - **Name**: `fld_Operator`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `119, 95`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `+`
7. Drag a second label field to the card and set the following properties:
    - **Name**: `fld_Equals`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `119, 151`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `=`
8. Click on the rectangular graphic in the toolbar palette, and then draw a rectangular graphic on your card. Next, set the following properties:
    - **Name**: `grc_Line`
    - **Opaque**: Keep this checked
    - foregroundColor: White

- backgroundColor: White
- **Width**: `178`
- **Height**: `4`
- **Location**: `177`, `124`

With all your objects on the card, your interface should look like the following screenshot:



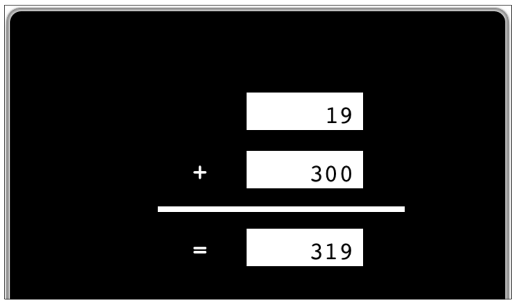
9. Add the following LiveCode script to the card:

```
on openCard
  put empty into fld "fld_nbr1"
  put empty into fld "fld_nbr2"
  put empty into fld "fld_nbr3"
end openCard
```

10. Add the following LiveCode script to the `fld_Equals` field:

```
on mouseUp
  local nbr1, nbr2, nbr3

  put the text of fld "fld_nbr1" into nbr1
  put the text of fld "fld_nbr2" into nbr2

  put (nbr1 + nbr2) into nbr3
  put nbr3 into fld "fld_nbr3"
end mouseUp
```

11. Test your application in the simulator or on an actual device. When you run your mobile application, enter `19` in the first field and `300` in the second, and select the equals sign. You should see results as shown in the following screenshot:



How it works...

For this recipe, we created two input text fields so that the user could enter two numbers that could be added together (summed). We added a script to the card using the `on openCard` handler to clear values from the three text fields. When the equals sign is selected, the two values are added using the plus (+) mathematical operator.

There's more...

We individually took the input values from the interface and put them into local variables (`nbr1` and `nbr2`). We then added the two numbers together, putting their sum into a third local variable, which is `nbr3`. Our last step was to put the value of `nbr3` into the `fld_nbr3` field. We could have simplified this with a single line of code:

```
put the text of fld "fld_nbr1" + the text of fld "fld_nbr2" into
  fld "fld_nbr3"
```

This line of code seems a bit long. So, in this recipe, we have broken down the code into component pieces, making it easier to read.

See also

- The *Subtracting numbers* recipe
- The *Multiplying numbers* recipe
- The *Dividing numbers* recipe
- The *Using advanced math* recipe
- The *Randomizing numbers* recipe

Subtracting numbers

Subtracting numbers using LiveCode is a simple task. To demonstrate how to subtract one number from another, we will create a user interface that accepts two numbers, and when the equals sign is selected, the results will be displayed.

How to do it...

Follow the steps in this recipe to create an interface and LiveCode script that subtracts one number from another:

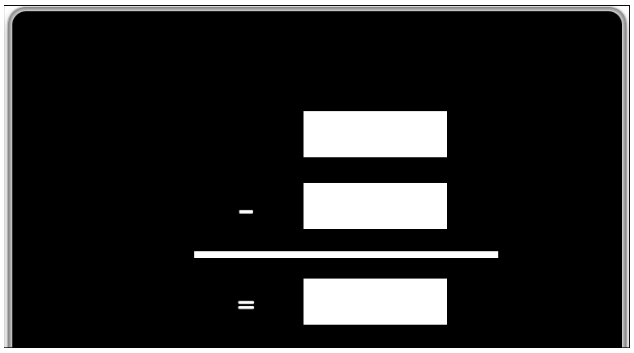
1. Open LiveCode and create a new main stack.
2. Set the background color of the default card to black.
3. Drag a new text entry field to the card and set the following properties:
    - **Name**: `fld_nbr1`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194, 53`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right
4. Drag a second text entry field to the card and set the following properties:
    - **Name**: `fld_nbr2`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194, 95`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right
5. Drag a third text entry field to the card and set the following properties:
    - **Name**: `fld_nbr3`
    - **Width**: `88`
    - **Height**: `31`

- **Location**: `194, 151`
- **Font**: **Courier**
- Text size: **18**
- Align text right

6. Drag a new label field to the card and set the following properties:
   - **Name**: `fld_Operator`
   - foregroundColor: White
   - **Width**: `46`
   - **Height**: `32`
   - **Location**: `119, 95`
   - **Font**: **Courier**
   - Text size: **18**
   - Text style: Bold
   - Align text center
   - **Contents**: -
7. Drag a second label field to the card and set the following properties:
   - **Name**: `fld_Equals`
   - foregroundColor: White
   - **Width**: `46`
   - **Height**: `32`
   - **Location**: `119, 151`
   - **Font**: **Courier**
   - Text size: **18**
   - Text style: Bold
   - Align text center
   - **Contents**: =
8. Click on the rectangular graphic in the toolbar palette, and then draw a rectangular graphic on your card. Next, set the following properties:
   - **Name**: `grc_Line`
   - **Opaque**: Keep this checked
   - foregroundColor: White

- backgroundColor: White
- **Width**: `178`
- **Height**: `4`
- **Location**: `177`, `124`

With all your objects on the card, your interface should look like the following screenshot:



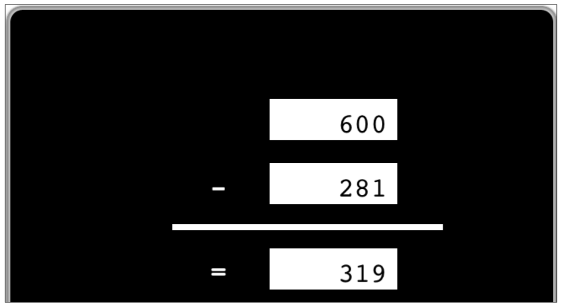
9. Add the following LiveCode script to the card:

```
on openCard
  put empty into fld "fld_nbr1"
  put empty into fld "fld_nbr2"
  put empty into fld "fld_nbr3"
end openCard
```

10. Add the following LiveCode script to the `fld_Equals` field:

```
on mouseUp
  local nbr1, nbr2, nbr3

  put the text of fld "fld_nbr1" into nbr1
  put the text of fld "fld_nbr2" into nbr2

  put (nbr1 - nbr2) into nbr3
  put nbr3 into fld "fld_nbr3"
end mouseUp
```

11. Test your application in the simulator or on an actual device. When you run your mobile application, enter `600` in the first field and `281` in the second, and select the equals sign. You should see results as shown in the following screenshot:



How it works...

For this recipe, we created two input text fields so that the user could enter two numbers—the second to be subtracted from the first. We added a script to the card using the `on openCard` handler to clear values from the three text fields. When the equals sign is selected, the second value is subtracted from the first using the minus (-) mathematical operator.

There's more...

We individually took the input values from the interface and put them into local variables (`nbr1` and `nbr2`). We then subtracted the second number from the first, putting the result into a third local variable, which is `nbr3`. Our last step was to put the value of `nbr3` into the `fld_nbr3` field. We could have simplified this with a single line of code:

```
put the text of fld "fld_nbr1" - the text of fld "fld_nbr2" into
  fld "fld_nbr3"
```

This line of code seems a bit long. So, in this recipe, we have broken down the code into component pieces, making it easier to read.

See also

- The *Adding numbers* recipe
- The *Multiplying numbers* recipe
- The *Dividing numbers* recipe
- The *Using advanced math* recipe
- The *Randomizing numbers* recipe

Multiplying numbers

Multiplying numbers using LiveCode is a simple task. To demonstrate how to multiply two numbers, we will create a user interface that accepts two numbers, and when the equals sign is selected, the product of the two numbers will be displayed.

How to do it...

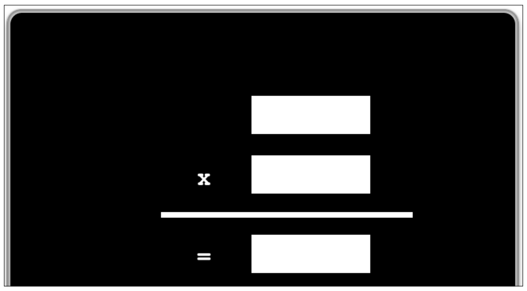
Follow the steps in this recipe to use LiveCode to multiply two numbers:

1. Open LiveCode and create a new main stack.
2. Set the background color of the default card to black.
3. Drag a new text entry field to the card and set the following properties:
    - **Name**: `fld_nbr1`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194, 53`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right
4. Drag a second text entry field to the card and set the following properties:
    - **Name**: `fld_nbr2`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194, 95`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right
5. Drag a third text entry field to the card and set the following properties:
    - **Name**: `fld_nbr3`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194, 151`
    - **Font**: **Courier**

- Text size: **18**
- Align text right

6. Drag a new label field to the card and set the following properties:
    - **Name**: `fld_Operator`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `119, 95`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `x`
7. Drag a second label field to the card and set the following properties:
    - **Name**: `fld_Equals`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `119, 151`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `=`
8. Click on the rectangular graphic in the toolbar palette, and then draw a rectangular graphic on your card. Next, set the following properties:
    - **Name**: `grc_Line`
    - **Opaque**: Keep this checked
    - foregroundColor: White
    - backgroundColor: White
    - **Width**: `178`
    - **Height**: `4`
    - **Location**: `177, 124`

With all your objects on the card, your interface should look like the following screenshot:



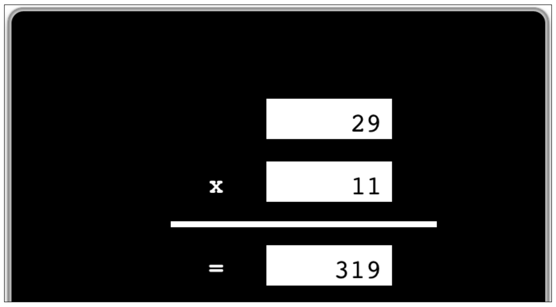
9. Add the following LiveCode script to the card:

```
on openCard
  put empty into fld "fld_nbr1"
  put empty into fld "fld_nbr2"
  put empty into fld "fld_nbr3"
end openCard
```

10. Add the following LiveCode script to the `fld_Equals` field:

```
on mouseUp
  local nbr1, nbr2, nbr3

  put the text of fld "fld_nbr1" into nbr1
  put the text of fld "fld_nbr2" into nbr2

  put (nbr1 * nbr2) into nbr3
  put nbr3 into fld "fld_nbr3"
end mouseUp
```

11. Test your application in the simulator or on an actual device. When you run your mobile application, enter `29` in the first field and `11` in the second, and select the equals sign. You should see results as shown in the following screenshot:



How it works...

For this recipe, we created two input text fields so that the user could enter two numbers to be multiplied. We added a script to the card using the `on openCard` handler to clear values from the three text fields. When the equals sign is selected, the two values are multiplied using the multiplication (`*`) mathematical operator.

> In computer science, the asterisk (`*`) is the mathematical operator for multiplication and not an "x".

There's more...

We individually took the input values from the interface and put them into local variables (`nbr1` and `nbr2`). We then multiplied the two numbers, putting their product into a third local variable, which is `nbr3`. Our last step was to put the value of `nbr3` into the `fld_nbr3` field. We could have simplified this with a single line of code:

```
put the text of fld "fld_nbr1" * the text of fld "fld_nbr2" into
  fld "fld_nbr3"
```

This line of code seems a bit long. So, in this recipe, we have broken down the code into component pieces, making it easier to read.

See also

- The *Adding numbers* recipe
- The *Subtracting numbers* recipe
- The *Dividing numbers* recipe
- The *Using advanced math* recipe
- The *Randomizing numbers* recipe

Dividing numbers

Dividing numbers using LiveCode is a simple task. To demonstrate how to divide numbers, we will create a user interface that accepts two numbers, and when the equals sign is selected, the result of division will be displayed.

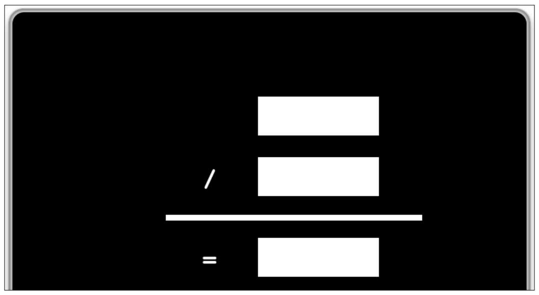
How to do it...

Follow the steps in this recipe to use the LiveCode script to divide numbers:

1. Open LiveCode and create a new main stack.
2. Set the background color of the default card to black.
3. Drag a new text entry field to the card and set the following properties:
    - **Name**: `fld_nbr1`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194,53`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right
4. Drag a second text entry field to the card and set the following properties:
    - **Name**: `fld_nbr2`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194,95`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right
5. Drag a third text entry field to the card and set the following properties:
    - **Name**: `fld_nbr3`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `194,151`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text right

6. Drag a new label field to the card and set the following properties:
   - **Name**: `fld_Operator`
   - foregroundColor: White
   - **Width**: `46`
   - **Height**: `32`
   - **Location**: `119`, `95`
   - **Font**: **Courier**
   - Text size: **18**
   - Text style: Bold
   - Align text center
   - **Contents**: `/`
7. Drag a second label field to the card and set the following properties:
   - **Name**: `fld_Equals`
   - foregroundColor: White
   - **Width**: `46`
   - **Height**: `32`
   - **Location**: `119`, `151`
   - **Font**: **Courier**
   - Text size: **18**
   - Text style: Bold
   - Align text center
   - **Contents**: `=`
8. Click on the rectangular graphic in the toolbar palette, and then draw a rectangular graphic on your card. Next, set the following properties:
   - **Name**: `grc_Line`
   - **Opaque**: Keep this checked
   - foregroundColor: White
   - backgroundColor: White
   - **Width**: `178`
   - **Height**: `4`
   - **Location**: `177`, `124`

With all your objects on the card, your interface should look like the following screenshot:



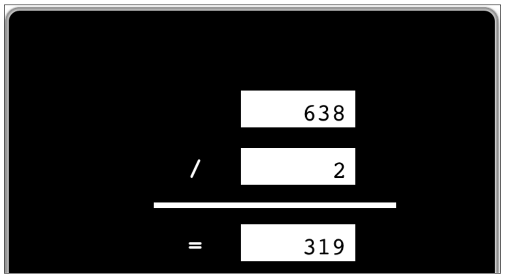
9. Add the following LiveCode script to the card:

```
on openCard
  put empty into fld "fld_nbr1"
  put empty into fld "fld_nbr2"
  put empty into fld "fld_nbr3"
end openCard
```

10. Add the following LiveCode script to the `fld_Equals` field:

```
on mouseUp
  local nbr1, nbr2, nbr3

  put the text of fld "fld_nbr1" into nbr1
  put the text of fld "fld_nbr2" into nbr2

  put (nbr1 / nbr2) into nbr3
  put nbr3 into fld "fld_nbr3"
end mouseUp
```

11. Test your application in the simulator or on an actual device. When you run your mobile application, enter `638` in the first field and `2` in the second, and select the equals sign. You should see results as shown in the following screenshot:



How it works...

For this recipe, we created two input text fields so that the user could enter two numbers to be divided. We added a script to the card using the `on openCard` handler to clear values from the three text fields. When the equals sign is selected, the two values are divided using the division (`/`) mathematical operator.

There's more...

We individually took the input values from the interface and put them into local variables (`nbr1` and `nbr2`). We then divided the two numbers (the first by the second), putting the results into a third local variable, which is `nbr3`. Our last step was to put the value of `nbr3` into the `fld_nbr3` field. We could have simplified this with a single line of code:

```
put the text of fld "fld_nbr1" / the text of fld "fld_nbr2" into
  fld "fld_nbr3"
```

This line of code seems a bit long. So, in this recipe, we have broken down the code into component pieces, making it easier to read.

See also

- The *Adding numbers* recipe
- The *Subtracting numbers* recipe
- The *Multiplying numbers* recipe
- The *Using advanced math* recipe
- The *Randomizing numbers* recipe

Using advanced math

In this recipe, you will learn how to use multiple math operations in a single line of LiveCode script. You will use operator precedence and nested operations. To facilitate this, we will use the common math trick where a person is asked to choose a number, and then perform the following operations on that number, the result always being `3`:

1. Choose a number.
2. Add 5.
3. Double the result.
4. Subtract 4.
5. Divide the result by 2.
6. Subtract the number you started with.

   The result is always 3

How to do it...

Follow the steps in this recipe to program a multiple-step mathematical operation:

1. Open LiveCode and create a new main stack.
2. Set the background color of the default card to black.
3. Drag a button to the card and set the following properties:
    - **Name**: `btn_GetNumber`
    - **Label**: `Choose a Number`
    - **Width**: `122`
    - **Height**: `23`
    - **Location**: `71`, `29`
    - threeD: Keep this unchecked
    - backgroundColor: White
4. Drag a new text entry field to the card and set the following properties:
    - **Name**: `fld_nbr1`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `276`, `23`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text center
    - traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
    - threeD: Keep this unchecked
    - backgroundColor: White
5. Drag a second text entry field to the card and set the following properties:
    - **Name**: `fld_nbr2`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `276, 64`
    - **Font**: **Courier**
    - Text size: **18**

- Align text center
- traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
- threeD: False
- backgroundColor: White

6. Drag a third text entry field to the card and set the following properties:
   - **Name**: `fld_nbr3`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `276, 104`
   - **Font**: **Courier**
   - Text size: **18**
   - Align text center
   - traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
   - threeD: Keep this unchecked
   - backgroundColor: White
7. Drag a fourth text entry field to the card and set the following properties:
   - **Name**: `fld_nbr4`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `276, 141`
   - **Font**: **Courier**
   - Text size: **18**
   - Align text center
   - traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
   - threeD: Keep this unchecked
   - backgroundColor: White
8. Drag a fifth text entry field to the card and set the following properties:
   - **Name**: `fld_nbr5`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `276, 179`
   - **Font**: **Courier**

- Text size: **18**
- Align text center
- traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
- threeD: Keep this unchecked
- backgroundColor: White

9. Drag a sixth text entry field to the card and set the following properties:
    - **Name**: `fld_nbr6`
    - **Width**: `88`
    - **Height**: `31`
    - **Location**: `276, 217`
    - **Font**: **Courier**
    - Text size: **18**
    - Align text center
    - traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
    - threeD: Keep this unchecked
    - backgroundColor: White
10. Drag a new label field to the card and set the following properties:
    - **Name**: `fld_Operator1`
    - foregroundColor: White
    - **Width**: `118`
    - **Height**: `26`
    - **Location**: `67, 66`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text left
    - **Contents**: `Add 5 (+5)`
11. Drag a second label field to the card and set the following properties:
    - **Name**: `fld_Operator2`
    - foregroundColor: White
    - **Width**: `160`
    - **Height**: `26`

- **Location**: `88, 104`
- **Font**: **Courier**
- Text size: **18**
- Text style: Bold
- Align text left
- **Contents**: `Double Result (x2)`

12. Drag a third label field to the card and set the following properties:
    - **Name**: `fld_Operator3`
    - foregroundColor: White
    - **Width**: `174`
    - **Height**: `26`
    - **Location**: `95, 141`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text left
    - **Contents**: `Subtract 4`
13. Drag a fourth label field to the card and set the following properties:
    - **Name**: `fld_Operator4`
    - foregroundColor: White
    - **Width**: `174`
    - **Height**: `26`
    - **Location**: `95, 178`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text left
    - **Contents**: `Divide by 2`
14. Drag a fifth label field to the card and set the following properties:
    - **Name**: `fld_Operator5`
    - foregroundColor: White
    - **Width**: `174`

- **Height**: `36`
- **Location**: `95, 215`
- **Font**: **Courier**
- Text size: **18**
- Text style: Bold
- Align text left
- **Contents**: `Subtract the number you started with`
- dontWrap: Keep this unchecked

15. Drag a sixth label field to the card and set the following properties:
    - **Name**: `fld_Explain`
    - foregroundColor: Yellow
    - **Width**: `218`
    - **Height**: `32`
    - **Location**: `171, 249`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text left
    - **Contents**: `The result is always 3`
16. Drag a seventh label field to the card and set the following properties:
    - **Name**: `fld_Equals1`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `209, 66`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `=`
17. Drag an eighth label field to the card and set the following properties:
    - **Name**: `fld_Equals2`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`

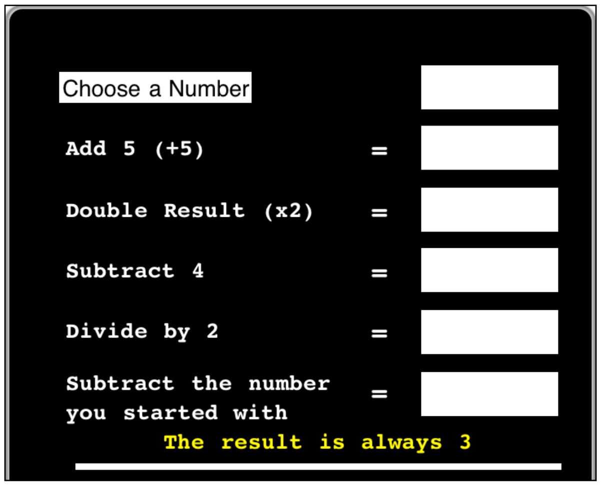
- **Location**: `209, 104`
- **Font**: **Courier**
- Text size: **18**
- Text style: Bold
- Align text center
- **Contents**: =

18. Drag a ninth label field to the card and set the following properties:
    - **Name**: `fld_Equals3`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `209, 141`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: =
19. Drag a tenth label field to the card and set the following properties:
    - **Name**: `fld_Equals4`
    - foregroundColor: White
    - **Width**: `46`
    - **Height**: `32`
    - **Location**: `209, 178`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: =
20. Drag an eleventh label field to the card and set the following properties:
    - **Name**: `fld_Equals5`
    - foregroundColor: White
    - **Width**: `46`

- **Height**: `32`
- **Location**: `209, 215`
- **Font**: **Courier**
- Text size: **18**
- Text style: Bold
- Align text center
- **Contents**: `=`

21. Click on the rectangular graphic in the toolbar palette, and then draw a rectangular graphic on your card. Next, set the following properties:
    - **Name**: `grc_Line`
    - **Opaque**: Keep this checked
    - foregroundColor: White
    - backgroundColor: White
    - **Width**: `301`
    - **Height**: `4`
    - **Location**: `172, 212`

    With all your objects on the card, your interface should look like the following screenshot:



22. Add the following LiveCode script to the card:

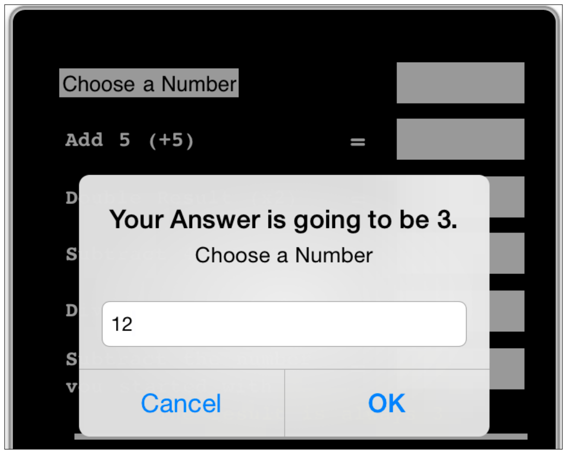
```
on openCard
  put empty into fld "fld_nbr1"
  put empty into fld "fld_nbr2"
```

```
    put empty into fld "fld_nbr3"
    put empty into fld "fld_nbr4"
    put empty into fld "fld_nbr5"
    put empty into fld "fld_nbr6"
  end openCard
```

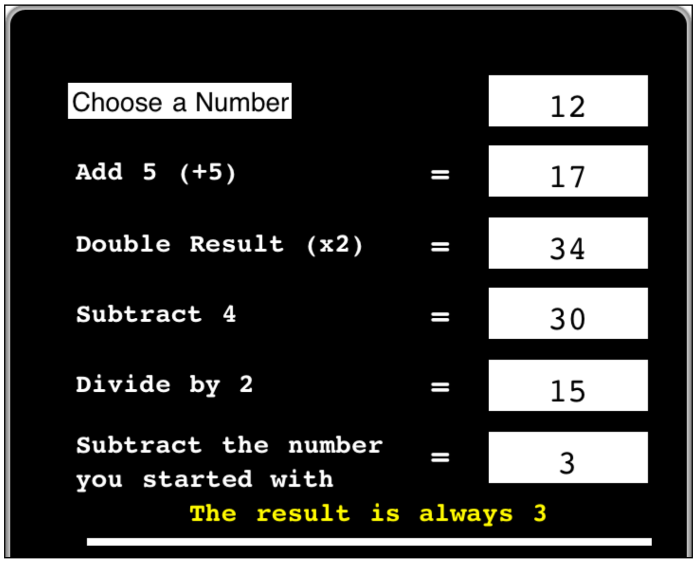
23. Add the following LiveCode script to the `btn_GetNumber` button:

```
  on mouseUp
    local tOriginal, tNbr

    ask "Choose a Number" titled "Your Answer is going to be
      3."
    put it into tOriginal
    put tOriginal into tNbr
    put tNbr into fld "fld_nbr1"
    --
    put tNbr + 5 into fld "fld_nbr2"
    --
    put (tNbr + 5) * 2 into fld "fld_nbr3"
    --
    put ((tNbr + 5) * 2) - 4 into fld "fld_nbr4"
    --
    put (((tNbr + 5) * 2) - 4) / 2 into fld "fld_nbr5"
    --
    put ((((tNbr + 5) * 2) - 4) / 2) - tOriginal into fld
      "fld_nbr6"
  end mouseUp
```

24. Test your application in the simulator or on an actual device. When you run your mobile application, you will see a pop up (shown in the following screenshot). Enter `12` and select **OK**.



You should see results as shown in the following screenshot:



```
Choose a Number                    12
Add 5 (+5)                   =     17
Double Result (x2)           =     34
Subtract 4                   =     30
Divide by 2                  =     15
Subtract the number          =      3
you started with
        The result is always 3
```

How it works...

For this recipe, we created a button that asked us for user input. Once the number is retrieved, we put that number into two local variables: one to store the original value (`tOriginal`), and the other for operational use. As we perform each step in the math trick, we do not modify the latest results; instead, we perform calculations from the beginning. To ensure that our calculations are accurate, we use open and close parentheses pairings to isolate each mathematical operation.

There's more...

You might be familiar with the order of operators or operator precedence from your early school years. Now is the time to brush up on that area of mathematics. Let's examine the mathematical equation of *10 + 6 / 3*. If we process the equation left to right we get *10 + 6 = 16* divided by *3*, which results in *5.333333*. This is the incorrect solution.

If we remember that we should divide before we add, we would divide *6* by *3* with a result of *2*, and then add it to *10* with a result of *12*. This is the correct solution.

It becomes very clear that we cannot simply perform calculations programmatically without specific knowledge of the order of precedence.

The following table shows you the proper operator order of precedence:

Precedence	**Name**	**Symbol**	**Explanation**
1	Grouping	( )	Expressions in parenthesis are evaluated first. When nested, the innermost dataset is evaluated first.
2	Unary	not bitNot there is a there is no	Unary operations act on a single operand only.
3	Exponent	^	Also referred to as the power of a number.
4	Multiplication/ Division	* / div mod	Each of these has the same order of precedence. If more than one is used in an equation, they are computed left to right.
5	Addition/ Subtraction	+ -	Both of these have the same order of precedence. If more than one is used in an equation, they are computed left to right.
6	Concatenation	& && ,	These are string operators (join strings).
7	Comparison	< > <= >= contains is/is not among is/is not in is/is not within is/is not a	These operators compare two values.

Precedence	Name	Symbol	Explanation
8	Equality	= is <> != is not	These operators compare two values for equality.
9	bitAnd	bitAnd	
10	bitXOr	bitXOr	
11	bitOr	bitOr	
12	And	and	
13	Or	or	
14	Function calls		This is the lowest priority operator.

See also

- The *Adding numbers* recipe
- The *Subtracting numbers* recipe
- The *Multiplying numbers* recipe
- The *Dividing numbers* recipe
- The *Randomizing numbers* recipe

Randomizing numbers

There can be a number of reasons you will want your mobile app to generate random numbers. You might create a slot or another gambling app or simply want enemy robots in a game to come from random vectors. Regardless of your need, LiveCode makes it easy to generate random numbers.

How to do it...

In this recipe, you will create a user interface that allows the user to enter upper and lower limits and then generate a random number within those limits.

1. Open LiveCode and create a new main stack.

2. Drag a new label field to the card and set the following properties:
    - **Name**: `fld_Lower`
    - foregroundColor: Yellow
    - **Width**: `158`
    - **Height**: `32`
    - **Location**: `87, 64`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `Set Lower Limit:`
3. Drag a second label field to the card and set the following properties:
    - **Name**: `fld_Upper`
    - foregroundColor: Yellow
    - **Width**: `158`
    - **Height**: `32`
    - **Location**: `259, 64`
    - **Font**: **Courier**
    - Text size: **18**
    - Text style: Bold
    - Align text center
    - **Contents**: `Set Upper Limit:`
4. Drag a new text entry field to the card and set the following properties:
    - **Name**: `fld_nbr1`
    - **Width**: `88`
    - **Height**: `31`

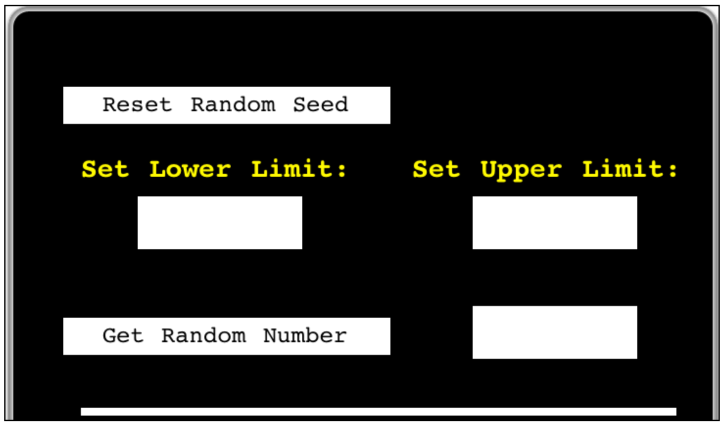
- **Location**: `87, 89`
- **Font**: **Courier**
- Text size: **18**
- Align text center
- backgroundColor: White

5. Drag a second text entry field to the card and set the following properties:
   - **Name**: `fld_nbr2`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `258, 89`
   - **Font**: **Courier**
   - Text size: **18**
   - Align text center
   - backgroundColor: White
6. Drag a third text entry field to the card and set the following properties:
   - **Name**: `fld_nbr3`
   - **Width**: `88`
   - **Height**: `31`
   - **Location**: `258, 145`
   - **Font**: **Courier**
   - Text size: **18**
   - Align text center
   - backgroundColor: White
   - traversalOn: Keep this unchecked (Keep **Focusable** unchecked)
7. Drag a new button to the card and set the following properties:
   - **Name**: `btn_Seed`
   - **Label**: `Reset Random Seed`
   - **Width**: `171`
   - **Height**: `23`
   - **Location**: `90, 29`
   - **Font**: **Courier**
   - Text size: **12**
   - Align text center
   - backgroundColor: White
   - threeD: Keep this unchecked

8. Drag a second button to the card and set the following properties:
    - **Name**: `btn_GetRandomNumber`
    - **Label**: `Get Random Number`
    - **Width**: `171`
    - **Height**: `23`
    - **Location**: `90, 147`
    - **Font**: **Courier**
    - Text size: **12**
    - Align text center
    - backgroundColor: White
    - threeD: Keep this unchecked

9. Click on the rectangular graphic in the toolbar palette, and then draw a rectangular graphic on your card. Next, set the following properties:
    - **Name**: `grc_Line`
    - **Opaque**: Keep it checked
    - foregroundColor: White
    - backgroundColor: White
    - **Width**: `311`
    - **Height**: `4`
    - **Location**: `171, 186`

    With all your objects on the card, your interface should look like the following screenshot:



10. Add the following LiveCode script to the card:

```
on openCard
  put empty into fld "fld_nbr1"
  put empty into fld "fld_nbr2"
  put empty into fld "fld_nbr3"
  --
  send "mouseUp" to btn "btn_Seed"
end openCard
```

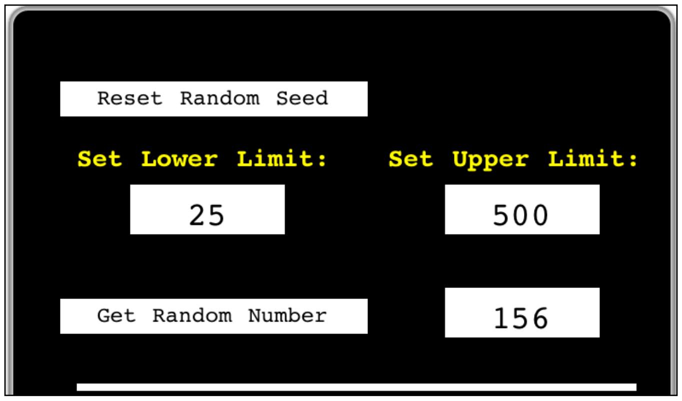
11. Add the following LiveCode script to the `btn_Seed` button:

```
on mouseUp
  set the randomSeed to the long seconds
end mouseUp
```

12. Add the following LiveCode script to the `btn_GetRandomNumber` button:

```
on mouseUp
  local lowerLimit, upperLimit

  put text of fld "fld_nbr1" into lowerLimit
  put text of fld "fld_nbr2" into upperLimit

  put random(upperLimit - lowerLimit + 1) + lowerLimit -1
    into fld "fld_nbr3"
end mouseUp
```

13. Test your application in the simulator or on an actual device. When you run your mobile application, you will need to enter both a lower and an upper limit. In the following screenshot, `25` was entered as a lower limit, `500` was entered as an upper limit, and the random number generated was **156**:



25

500

156

How it works...

Our code in this recipe accomplished four things. First, when the card is opened for the first time, the `fld_nbr1`, `fld_nbr2`, and `fld_nbr3` fields are cleared. Our `on openCard` handler also sends the `mouseUp` message to our `btn_Seed` button. This ensures that the random seed is fresh each time the card is opened. Secondly, we created a `btn_Seed` button so that the user can generate a new random seed whenever they choose to. Thirdly, we allow the user to input both upper and lower limit values. Lastly, we calculate a random number between the upper and lower limits.

There's more...

A random seed is a number that is randomly generated and is used to instantiate a new random number. It is important to use a unique random seed each time you need to generate a random number. While LiveCode uses a new random seed each time the application is started, it is a good habit to always generate a new random seed each time you need a random number. Using a function such as `long seconds` is a good technique. The `long seconds` function returns the number of seconds since midnight, January 1, 1970 (GMT).

See also

- The *Adding numbers* recipe
- The *Subtracting numbers* recipe
- The *Multiplying numbers* recipe
- The *Dividing numbers* recipe
- The *Using advanced math* recipe

Opening a web page

There could be a need for you to open a web page within your mobile app. You are not likely to create a new mobile browser using LiveCode; however, LiveCode does provide basic functionality for you to be able to display web pages in your mobile apps. In this recipe, you will learn how to open and display a web page within your mobile application using LiveCode.

How to do it...

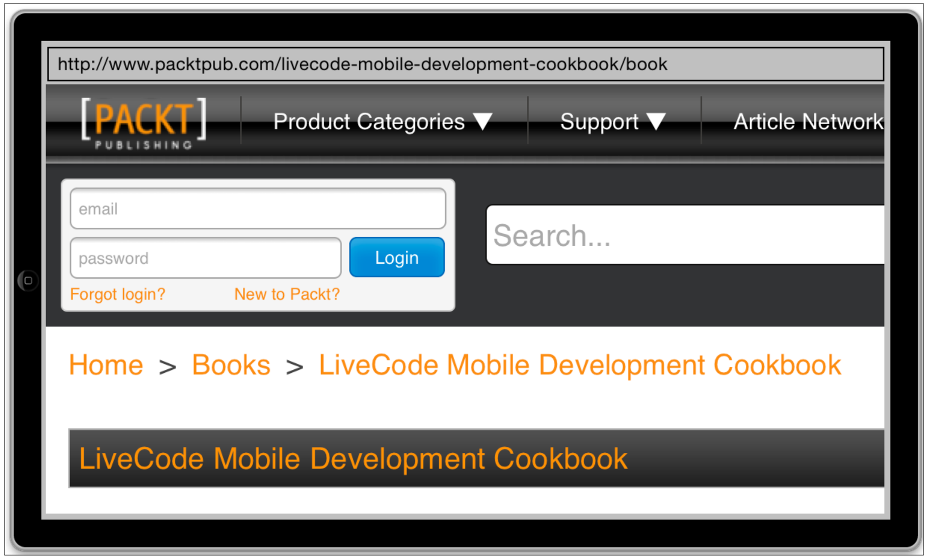
Follow the steps in this recipe to create a mobile app that opens a web page:

1. Open LiveCode and create a new main stack.
2. Using the Standalone Application Settings dialog window, enable the landscape rotation of your selected mobile device.

3. Create a group on your card with the following specifications:
   - **Name**: `Browser`
   - **Width**: `312`
   - **Height**: `390`
   - **Location**: `160,225`
4. Create a text input field on your card with the following specifications:
   - **Name**: `fld_URL`
   - **Width**: `312`
   - **Height**: `23`
   - **Location**: `160,15`
5. Add the following code to the card:

```
local browserID

on preOpenCard
  mobileControlCreate "browser"
  put the result into browserID

  mobileControlSet browserID, "visible", "true"
  mobileControlSet browserID, "url",
    "http://www.packtpub.com/livecode-mobile-development-
      cookbook/book"
  resizeStack
end preOpenCard

on closeCard
  mobileControlDelete browserID
end closeCard

on resizeStack
  set the rect of field "fld_URL" to the left of field
    "fld_URL", the top of field "fld_URL", the width of
    this card - 4, the bottom of field "fld_URL"
  set the rect of group "Browser" to the left of group
    "Browser", the top of group "Browser", the width of
    this card - 4, the height of this card - 4
  mobileControlSet browserID, "rect", the rect of group
    "Browser"
end resizeStack

on browserFinishedLoading pUrl
  put pUrl into fld "fld_URL"
end browserFinishedLoading
```

6. Test your application in the simulator or on an actual device. You should see the Packt Publishing website displayed as shown in the following screenshot. If the website you have displayed differs a bit, that is okay. The content and layout of websites change frequently.

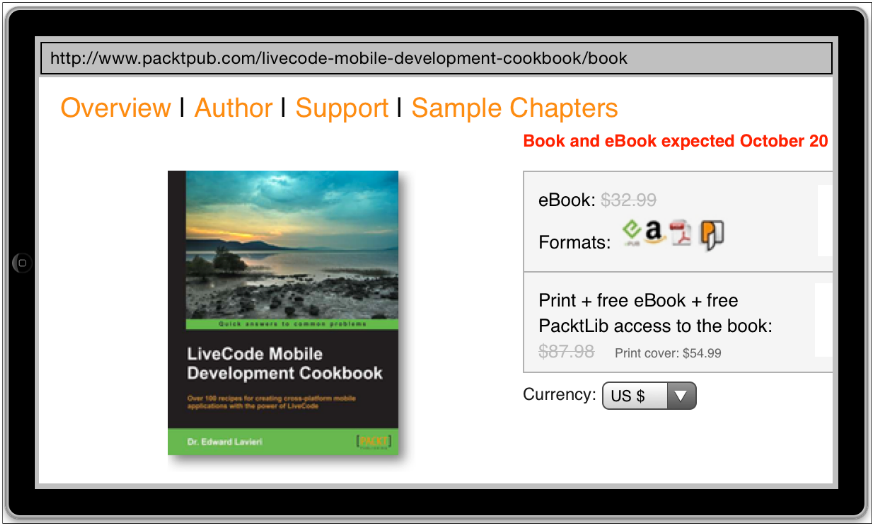


http://www.packtpub.com/livecode-mobile-development-cookbook/book

Home > Books > LiveCode Mobile Development Cookbook

LiveCode Mobile Development Cookbook

7. If you are using a simulator to test your mobile application, click on the screen to scroll up and down. Otherwise, use your finger to scroll. As you can see from the following screenshot, the entire web page is loaded. If the website you have displayed differs a bit, that is okay. The content and layout of websites change frequently.



http://www.packtpub.com/livecode-mobile-development-cookbook/book

Overview | Author | Support | Sample Chapters
LiveCode Mobile Development Cookbook
Dr. Edward Lavieri
Formats:
Currency: US $

Book and eBook expected October 20

eBook: ~~$32.99~~

Print + free eBook + free PacktLib access to the book: ~~$87.98~~ Print cover: $54.99

How it works...

We started this recipe by creating a local variable named `browserID`. This provides us with a reference ID for the browser instantiation. Using the `on preOpenCard` handler, we created a browser control (`mobileControlCreate "browser"`) and put the result into our `browserID` variable. Also, in the `preOpenCard` handler, we set the browser control to visible (`mobileControlSet browserID, "visible", "true"`), set the initial URL (`mobileControlSet browserID, "url", "http://www.packtpub.com/livecode-mobile-development-cookbook/book"`), and made a call to the `resizeStack` handler.

The `resizeStack` handler does a good job of resizing the `fld_URL` field and the **Browser** group. This is especially nice to use when multiple orientations (portrait and landscape) are supported.

The `browserFinishedLoading` message is sent once the web page has been loaded. So, our handler puts the loaded URL in the `fld_URL` field.

The last thing our code does is that it deletes the instantiated `browserID` as a good memory steward using the `on closeCard` handler.

See also

- The *Querying web data* recipe

Querying web data

In this recipe, we will load a web page into the memory and search for specific text to be displayed. Specifically, we will load this book's web page to the Packt Publishing website and find the *Who this book is for* section.

How to do it...

Follow the steps in this recipe to pull data from a web page, format it, and display it to the user in your mobile app:

1. Open LiveCode and create a new main stack.
2. Set the background color of the stack's card to black.
3. Drag a button to the card and set the following properties using the property inspector:
    - **Name**: `btn_Start`
    - **Label**: `Start`
    - **Width**: `82`
    - **Height**: `23`

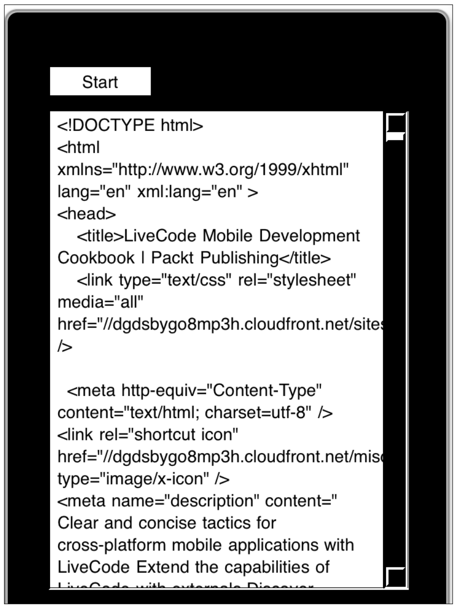
- **Location**: `57, 37`
- backgroundColor: White
- threeD: Keep this unchecked
- border: Keep this unchecked

4. Drag a scrolling field to the card and set the following properties using the property inspector:
   - **Name**: `fld_output`
   - **Width**: `294`
   - **Height**: `392`
   - **Location**: `161, 256`
   - backgroundColor: White

5. Add the following code to the `btn_Start` button:

```
on mouseUp
  local pageData, tLines
  put empty into fld "output"
  put URL "http://www.packtpub.com/livecode-mobile-
    development-cookbook/book" into fld "output"
end mouseUp
```

6. Test the app in a simulator or on an actual device. What you will see (as shown in the following screenshot) is that the text of the web page is displayed. This does us little good. In the following steps, we will call out the specific text we are looking for.

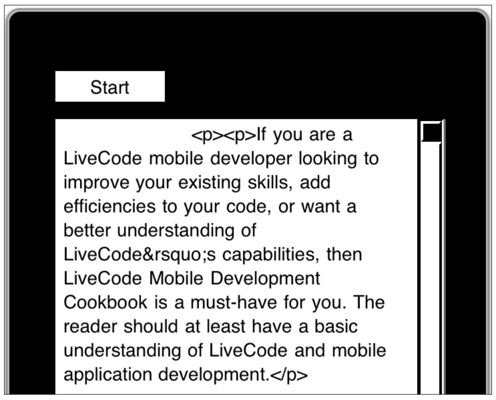


```
<!DOCTYPE html>
<html
xmlns="http://www.w3.org/1999/xhtml"
lang="en" xml:lang="en" >
<head>
   <title>LiveCode Mobile Development
Cookbook | Packt Publishing</title>
   <link type="text/css" rel="stylesheet"
media="all"
href="//dgdsbygo8mp3h.cloudfront.net/sites
/>

  <meta http-equiv="Content-Type"
content="text/html; charset=utf-8" />
<link rel="shortcut icon"
href="//dgdsbygo8mp3h.cloudfront.net/misc
type="image/x-icon" />
<meta name="description" content="
Clear and concise tactics for
cross-platform mobile applications with
LiveCode Extend the capabilities of
```

7. Next, we will search for the `Who this book is for` text. Change the `btn_Start` button's code to the following:

```
on mouseUp
  local pageData, tLines
  put empty into fld "output"
  put URL "http://www.packtpub.com/livecode-mobile-
    development-cookbook/book" into pageData
  put the number of lines of pageData into tLines
  repeat with i = 1 to tLines
    If line i of pageData contains "Who this book is for"
      then
      put line i +2 of pageData into fld "output"
exit mouseUp
    end if
  end repeat
end mouseUp
```

8. Test your application in the simulator or on an actual device. You will see that we have the right text, but it still needs to be cleaned up a bit, as shown in the following screenshot:



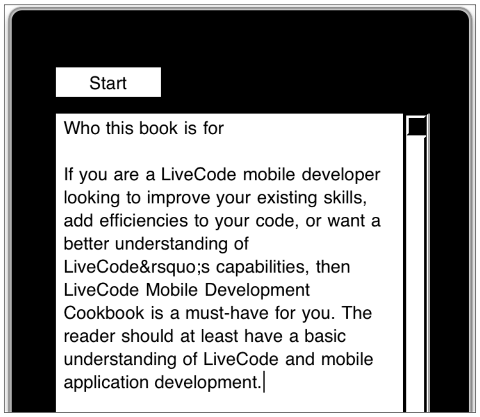
<p><p>If you are a LiveCode mobile developer looking to improve your existing skills, add efficiencies to your code, or want a better understanding of LiveCode&rsquo;s capabilities, then LiveCode Mobile Development Cookbook is a must-have for you. The reader should at least have a basic understanding of LiveCode and mobile application development.</p>

9. Next, we'll add scripting to clean up the text, and then we will display the results. Change the `btn_Start` button's code to the following:

```
on mouseUp
  local pageData, tLines
  put empty into fld "output"
  put URL "http://www.packtpub.com/livecode-mobile-
    development-cookbook/book" into pageData
  put the number of lines of pageData into tLines
  repeat with i = 1 to tLines
```

```
      If line i of pageData contains "Who this book is for"
        then
        put line i +2 of pageData into fld "output"
        cleanItUp
        exit mouseUp
      end if
    end repeat
  end mouseUp

  command cleanItUp
    local tText
    put the text of fld "output" into tText
    put empty into fld "output"

    put "Who this book is for" into line 1 of fld "output"
    replace "<p>" with "" in tText
    replace "</p>" with "" in tText
    replace tab with "" in tText
    put tText into line 3 of fld "output"
  end cleanItUp
```

10. Test your application in the simulator or on an actual device. As shown in the following screenshot, the text still needs to be cleaned up a bit. You'll want to add necessary code to remove unwanted text.



Who this book is for

If you are a LiveCode mobile developer looking to improve your existing skills, add efficiencies to your code, or want a better understanding of LiveCode&rsquo;s capabilities, then LiveCode Mobile Development Cookbook is a must-have for you. The reader should at least have a basic understanding of LiveCode and mobile application development.

How it works...

In this recipe, we queried a specific website that we were familiar with. This means that we knew what type of content we were looking for and how it was presented on the web page. This allowed us to load the web page data into the memory, and then keep only the section we were interested in.

See also

- The *Opening a web page* recipe

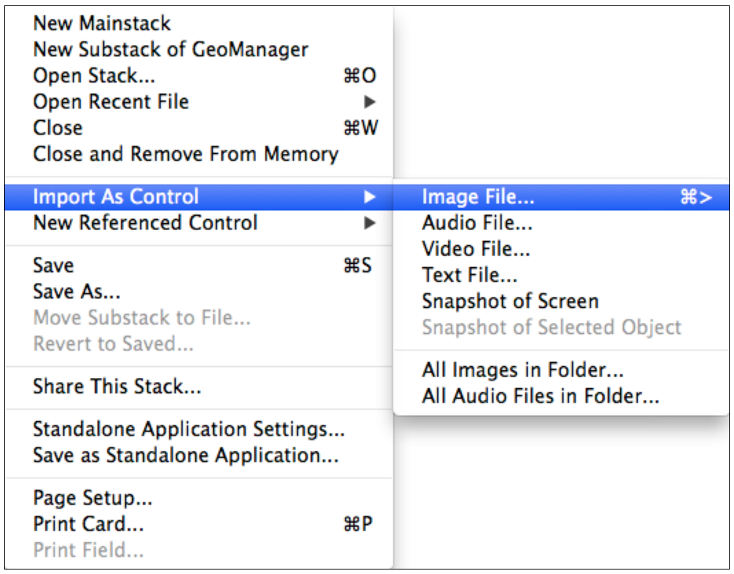
Using the geometry manager

In this recipe, you will learn about and gain experience with LiveCode's geometry manager. Specifically, you will load an image to a new stack and create links between the image and the right edge of the stack. You will select how the image behaves as the stack is resized.

How to do it...

Follow the steps in this recipe to learn how to use the geometry manager to enhance your mobile apps:

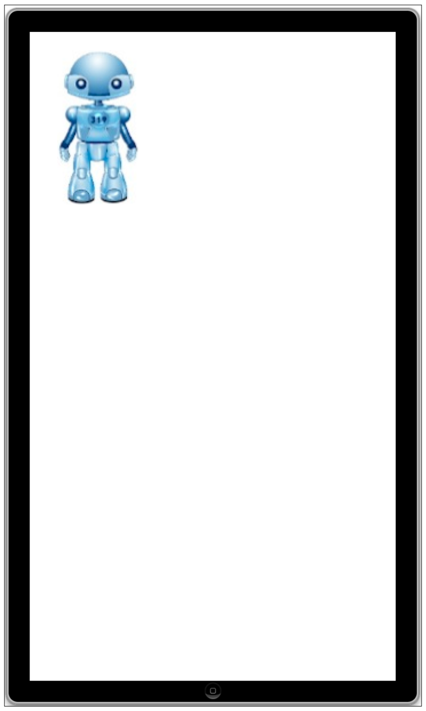
1. Open LiveCode and create a new main stack.
2. Set the background color of the default card to white.
3. Add an image to the card by selecting the **File** pull-down menu, then select **Import As Control**, and finally, select **Image File...**, as shown in the following screenshot:



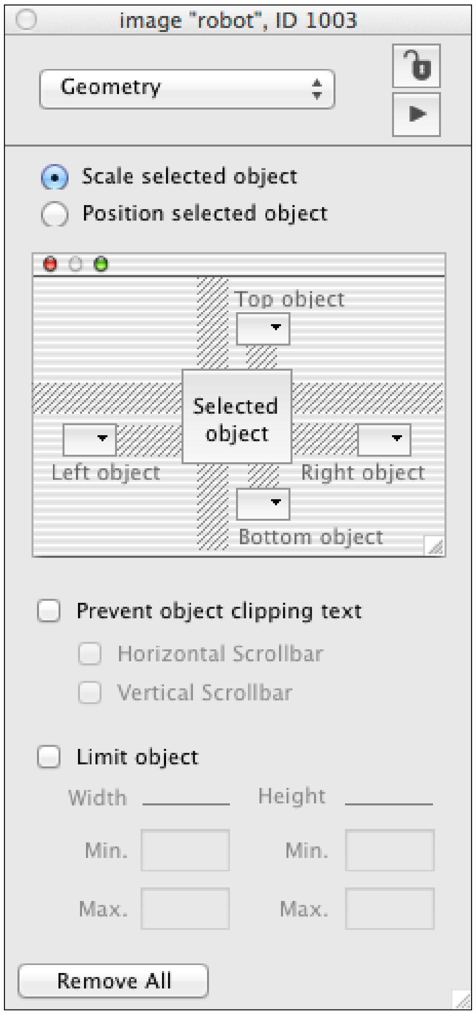
4. Select the `robot.png` file from your filesystem.

> The `robot.png` image file can be downloaded from this book's page on the Packt Publishing website (`www.packtpub.com`).

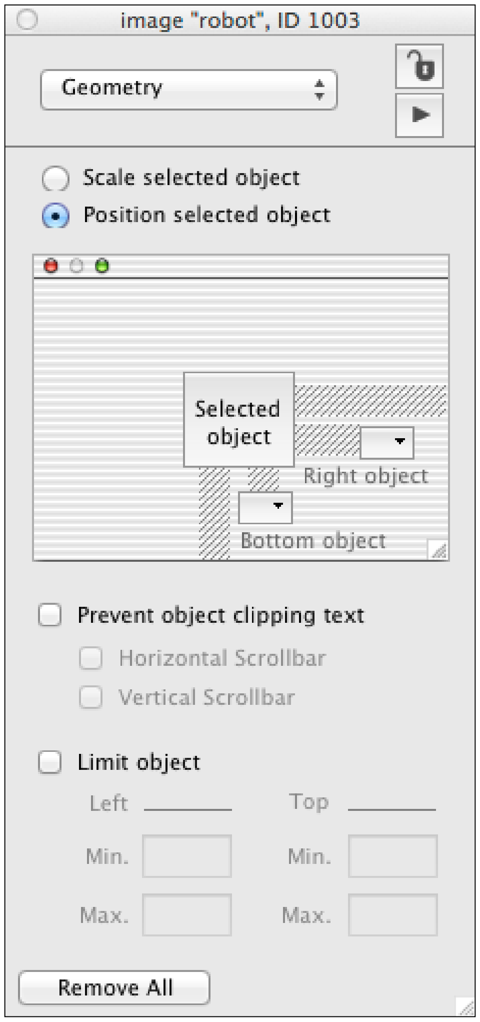
5. Change the name of the imported image to `robot`.
6. Set the location of the `robot` image to `59, 83`.
7. Test the app in a simulator or on an actual device. You will see the robot, as shown in the following screenshot:



8. With the `robot` image selected, select **Geometry** from the property inspector's pull-down menu as shown in the following screenshot:



9. As shown in the previous screenshot, there are three main functions of the **Geometry** interface: scale or position the selected object, prevent object-clipping text, and limit the object. The first set of radio buttons allows us to select how we want the image to behave when a stack is resized. The first option is to have the scale adjusted, and the second is to have the position adjusted. The default is to have the position adjusted. Select the **Position selected object** radio button so that the image's scale will not be adjusted; rather, the position will be adjusted when the stack is resized, as shown in the following screenshot:

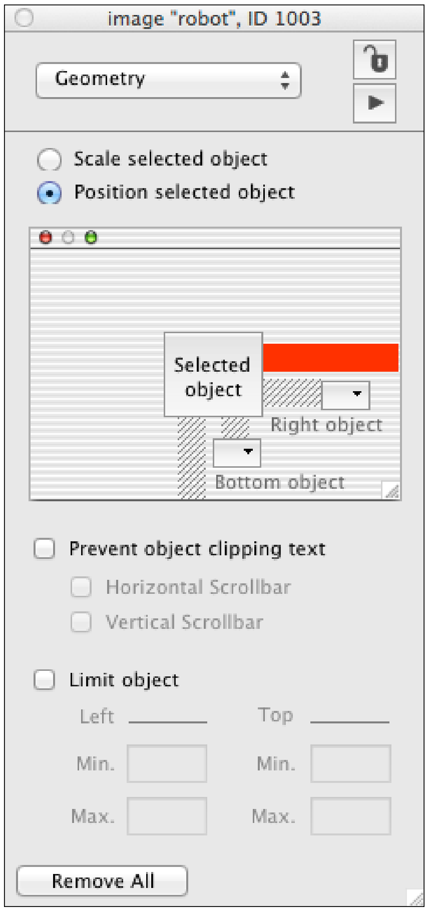


( ) Scale selected object
(•) Position selected object

[ ] Prevent object clipping text
[ ] Horizontal Scrollbar
[ ] Vertical Scrollbar

[ ] Limit object

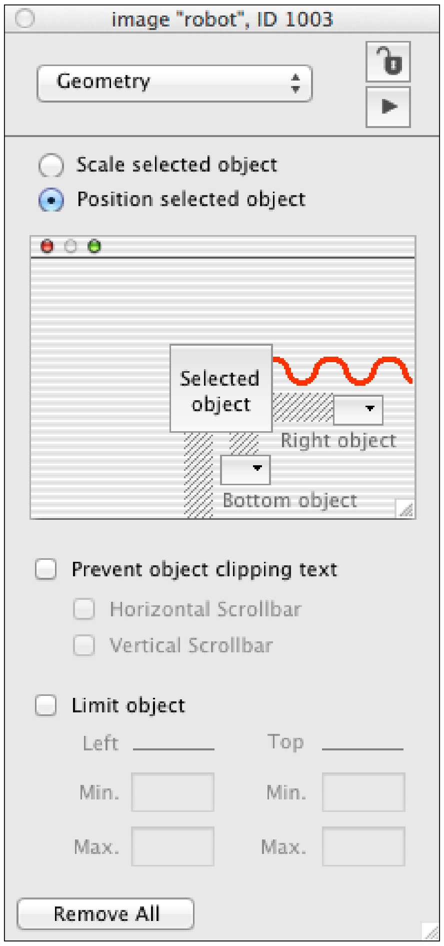
10. Next, we will click on the gray bar that extends to the right from the right-hand side of the box labeled **Selected object** in the geometry manager, as shown in the following screenshot. This results in the image remaining a fixed number of pixels from the right edge of the stack, regardless of what dimensions the stack is resized to.



11. Test the application in the IDE on your desktop. Resize the stack manually to illustrate the effects of the previous step.

You can resize stacks and cards programmatically or manually in the **IDE** (short for **Integrated Development Environment**). For this recipe, it is much quicker to test the resizing manually in the IDE.

12. Next, in the geometry manager, click on the solid bar that is extended from the box labeled **Selected object**. Now, the line should be wavy, as shown in the following screenshot. This results in the image being moved relative to the size of the stack as it is resized. In other words, the image will remain a specific percentage of the card's total width from the edge of the card.



13. Test the application in the IDE to see the effects of the previous step.

How it works...

The geometry manager is a powerful tool that can be used when your stack size changes at runtime. While this might not be common for mobile apps, it is an important functionality that one should know of.

There's more...

LiveCode's geometry manager also lets us assign scaling and positioning links between two objects and not just between an object and the stack/card boundary. In the geometry manager, there are drop-down menus (see the following screenshot) that allow you to select from objects on the current card.

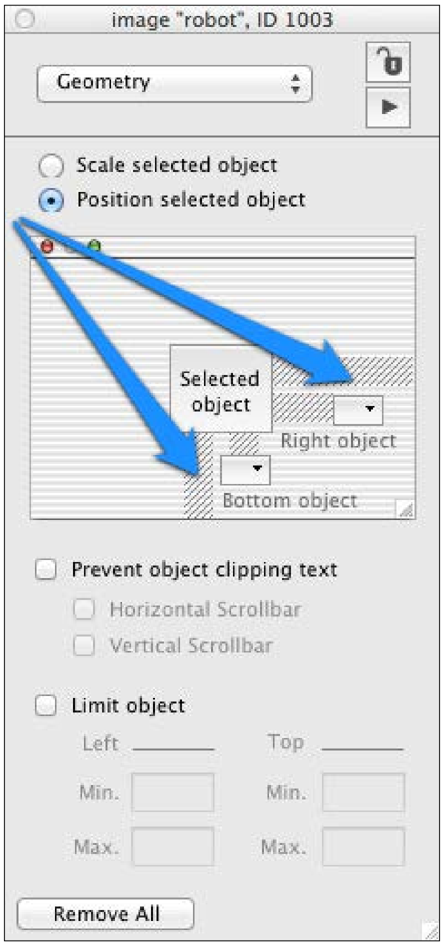


image "robot", ID 1003

Geometry

( ) Scale selected object
(•) Position selected object

[ ] Prevent object clipping text
[ ] Horizontal Scrollbar
[ ] Vertical Scrollbar

[ ] Limit object

Left		Top	
Min.		Min.	
Max.		Max.	

Remove All

Using invisible objects

Objects in LiveCode have the ability to be on screen but not visible. We can create invisible objects so that they provide a layer above underlying objects. This can benefit our efforts to develop mobile apps in several ways in order to include providing a method of capturing user input and the ability to add a script to the invisible object instead of multiple, underlying objects.

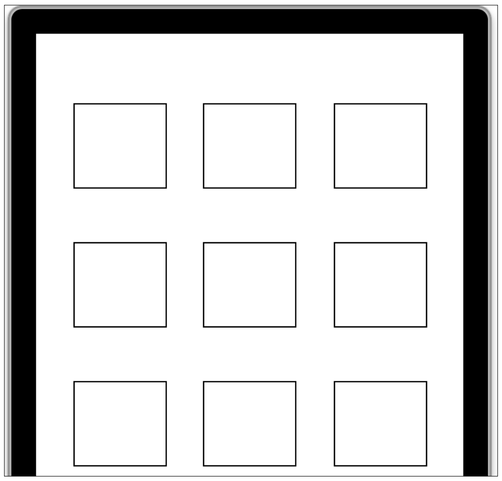
How to do it...

Follow the steps in this recipe to create an invisible object and assign the LiveCode script to it:

1. Open LiveCode and create a new main stack.
2. Set the background color of the stack to white.
3. Add nine rectangles to the stack with the following basic properties:
   - **Width**: `70`
   - **Height**: `64`
   - **Opaque**: Keep it checked
4. Name and position the nine rectangles as indicated in the following table:

Rectangle name	Location
`rect1`	`63,84`
`rect2`	`160,84`
`rect3`	`258,84`
`rect4`	`63,188`
`rect5`	`160,188`
`rect6`	`258,188`
`rect7`	`63,292`
`rect8`	`160,292`
`rect9`	`258,292`

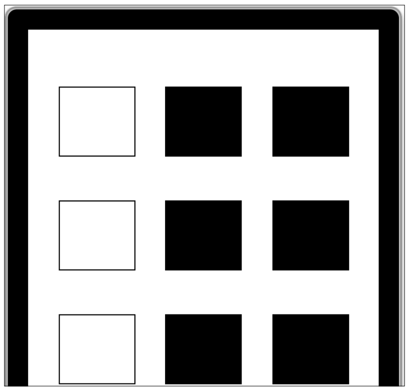
You should now see nine identical rectangles aligned as shown in the following screenshot:



5. Next, we will create an invisible object that will overlap six of the nine rectangles. Create a rectangle with the following properties:
   - **Name**: `invisible`
   - **Width**: `180`
   - **Height**: `292`
   - **Location**: `210,190`
   - **Opaque**: Keep this checked
   - **Blend Level**: 100 percent
6. Add the following script to the **invisible** graphic:

```
on mouseUp
  set the backgroundColor of grc "rect2" to black
  set the backgroundColor of grc "rect3" to black
  set the backgroundColor of grc "rect5" to black
  set the backgroundColor of grc "rect6" to black
  set the backgroundColor of grc "rect8" to black
  set the backgroundColor of grc "rect9" to black
end mouseUp
```

7. Test your application in the simulator or on an actual device. As illustrated in the following screenshot, once the **invisible** button is selected, the underlying six rectangles have their background color changed from white to black:



How it works...

Setting the blend level to 100 percent allows an object to be present on the screen but not seen. This is different from setting the `visible` property of an object to `false`. In that case, the object would not be available for user interaction.

Taking snapshots of a card

LiveCode is capable of capturing screen images of a stack. This means that you can capture what your LiveCode mobile app displays on the screen. This can be useful for allowing the user to take screenshots of in-app accomplishments such as with a drawing app. The number of uses of this type of functionality is seemingly limitless. In this recipe, you will learn how to create an image by taking a snapshot of a stack.

How to do it...

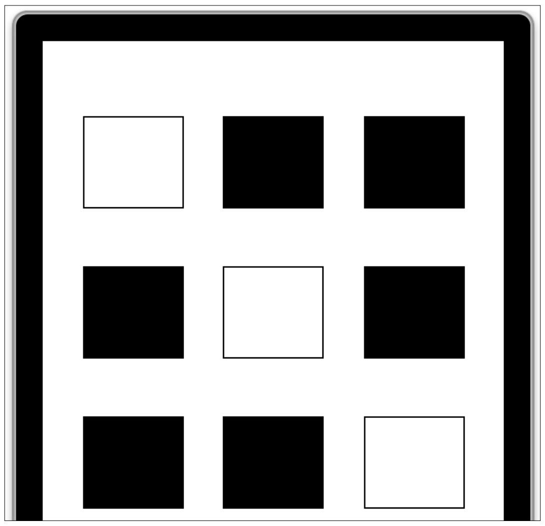
Follow the steps in this recipe to create a LiveCode script that creates a snapshot of the screen:

1. Open LiveCode and create a new main stack.
2. Set the background color of the stack to white.
3. Add nine rectangles to the stack with the following basic properties:
    - **Width**: `70`
    - **Height**: `64`
    - **Opaque**: Keep this checked

4. Name and position the nine rectangles as indicated in the following table:

Rectangle name	**backgroundColor**	**Location**
`rect1`	White	`63,84`
`rect2`	Black	`160,84`
`rect3`	Black	`258,84`
`rect4`	Black	`63, 188`
`rect5`	White	`160, 188`
`rect6`	Black	`258,188`
`rect7`	Black	`63,292`
`rect8`	Black	`160,292`
`rect9`	White	`258,292`

You should now see nine identical rectangles aligned as shown in the following screenshot:



5. With the following line of code, you can create an image of the screen:

```
import snapshot
```

How it works...

The `import snapshot` command can create an image from a screen snapshot. This command places the image in the center of the current card. It can be easily missed since the image is visually the same as the card beneath it. You then have the flexibility to do what you want to with the new image.

There's more...

The `import snapshot` command is capable of creating an image from a specific portion of the screen by assigning additional parameters to the command. This will be covered in the following recipe.

See also

- The *Taking snapshots of an area on a card* recipe

Taking snapshots of an area on a card

LiveCode is capable of capturing an image of a specific area of a stack. This can be useful when creating certain mobile apps. The number of uses for this type of functionality is seemingly limitless. In this recipe, you will learn how to create an image of a specific area of a stack.

How to do it...

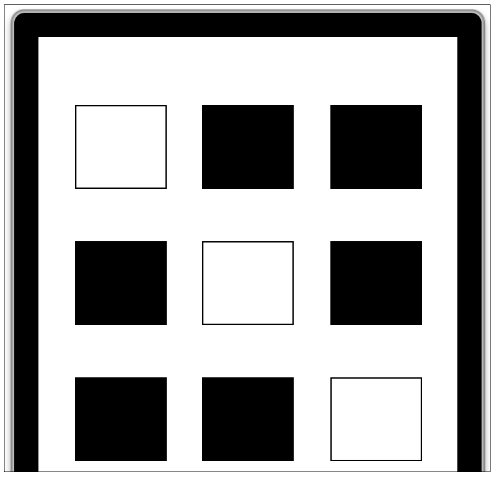
Follow the steps in this recipe to create an image by taking a screenshot of a specific area of a stack:

1. Open LiveCode and create a new main stack.
2. Set the background color of the stack to white.
3. Add nine rectangles to the stack with the following basic properties:
    - **Width**: `70`
    - **Height**: `64`
    - **Opaque**: Keep this checked

4. Name and position the nine rectangles as indicated in the following table:

Rectangle name	backgroundColor	Location
`rect1`	White	`63,84`
`rect2`	Black	`160,84`
`rect3`	Black	`258, 84`
`rect4`	Black	`63,188`
`rect5`	White	`160,188`
`rect6`	Black	`258,188`
`rect7`	Black	`63,292`
`rect8`	Black	`160,292`
`rect9`	White	`258,292`

You should now see nine identical rectangles aligned as shown in the following screenshot:



5. With the following line of code, you can create an image that encompasses the last two columns of rectangles:

```
import snapshot from rectangle 120,44,300,336 of this card
```

How it works...

The `import snapshot` command can create an image by taking a snapshot of the entire screen or a part of a screen. You then have the flexibility to do what you want with the created image.

See also

- The *Taking snapshots of a card* recipe

Detecting the operating system

In this recipe, you will learn how to determine what operating system your user has installed on their device.

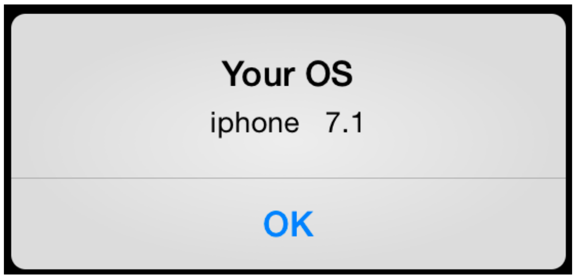
How to do it...

Perform the following steps to detect the installed operating system:

1. Open LiveCode and create a new main stack.
2. Add the following code to the stack's card:

```
on preOpenCard
  local tPlatform, tVersion

  put the platform into tPlatform
  put the systemVersion into tVersion

  answer tPlatform & tab & tVersion titled "Your OS"
end preOpenCard
```

3. Test your application in the simulator or on an actual device. See the following screenshot for a sample result:



Your OS

iphone   7.1

How it works...

In this recipe, we used two functions to determine the user's device platform and operating system version. The first function, `platform`, will return either `iphone` or `android`, depending on the mobile device being used. The second function, `systemVersion`, returns the numeric and decimal (that is 7.0.1) version of the user's OS.

Index