Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™



Professional Flash® Lite® Mobile Development

Build Flash Applications for Mobile Devices

Jermaine G. Anderson

www.it-ebooks.info



Programmer to Programmer™

Get more out of wrox.com

Interact

Take an active role online by participating in our P2P forums @ p2p.wrox.com

Wrox Online Library

Hundreds of our books are available online through Books24x7.com

Wrox Blox

Download short informational pieces and code to keep you up to date and out of trouble!

Join the Community

Sign up for our free monthly newsletter at newsletter.wrox.com

Browse

Ready for more Wrox? We have books and e-books available on .NET, SQL Server, Java, XML, Visual Basic, C#/ C++, and much more!

Contact Us.

We always like to get feedback from our readers. Have a book idea?
Need community support? Let us know by e-mailing wrox-partnerwithus@wrox.com

www.it-ebooks.info

PROFESSIONAL FLASH® LITE® MOBILE DEVELOPMENT

INTRODUCTION xvii

CHAPTER 1	Flash Is Mobile	1
CHAPTER 2	Getting Started	13
CHAPTER 3	Object-Oriented Programming for Flash Lite Development	33
CHAPTER 4	UI Components	75
CHAPTER 5	PureMVC ActionScript 2.0 Framework	107
CHAPTER 6	Creating a TV Listings Application	139
CHAPTER 7	Creating a Media Console	179
CHAPTER 8	Creating an Image Viewer Client	241
CHAPTER 9	Creating a Twitter Client	285
CHAPTER 10	Using Nokia's S60 Platform Services API	337
CHAPTER 11	Creating a Weather Client	385
CHAPTER 12	Using Sony Ericsson's Project Capuchin Platform Services API	439
CHAPTER 13	Packaging Flash Lite Applications for Distribution	485
APPENDIX	Flash Lite 3.x ActionScript 2.0 Quick Reference: From Array to XMLSocket	509

INDEX 537

www.it-ebooks.info

PROFESSIONAL

Flash® Lite® Mobile Development

Jermaine G. Anderson



Wiley Publishing, Inc.

www.it-ebooks.info

Professional Flash® Lite® Mobile Development

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-54748-9

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Control Number: 2009939787

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Flash Lite is a registered trademark of Adobe Systems Incorporated. All other trademarks are the property of their respective owners. Wiley Publishing, Inc. is not associated with any product or vendor mentioned in this book.

www.it-ebooks.info

CREDITS

ACQUISITIONS EDITOR
Scott Meyers

PROJECT EDITOR
Kenyon Brown

TECHNICAL EDITOR
Darren Osadchuk

PRODUCTION EDITOR
Kathleen Wisor

COPY EDITORS
Eric Newman
Foxxe Editorial Services

EDITORIAL DIRECTOR
Robyn B. Siesky

EDITORIAL MANAGER
Mary Beth Wakefield

MARKETING MANAGER
David Mayhew

PRODUCTION MANAGER
Tim Tate

VICE PRESIDENT AND EXECUTIVE GROUP PUBLISHER
Richard Swadley

VICE PRESIDENT AND EXECUTIVE PUBLISHER
Barry Pruett

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Lynsey Stanford

PROOFREADER
Nate Pritts, Word One

INDEXER
Robert Swanson

COVER DESIGNER
Mike Trent

COVER IMAGE
iStockPhoto

www.it-ebooks.info

ABOUT THE AUTHOR



JERMAINE G. ANDERSON has been programming for the past 10 years, specializing in Flash and, more recently, the Flex framework. His work predominantly centers on video streaming, and producing on-demand and live video content for the sports and entertainment industries where he has created web components and rich consumer products using Adobe Flash, Flex, and AIR. In 2004, he became interested in Flash mobile applications; this was intensified after winning the "Best Productivity" category for his mobile TV guide concept in the first ever Macromedia Flash Lite 1.1 contest in 2005. He has since spent time working for UK mobile phone operator T-Mobile, where he created dynamic user interfaces and prototypes using Flash Lite. He has managed an international team of developers and now works for leading entertainment and communications company British Sky Broadcasting. He holds a BSc (Hons) in Chemistry from the University of Birmingham, and an MSc in Computer Studies from Sheffield Hallam University.

www.it-ebooks.info

ACKNOWLEDGMENTS

WRITING THIS BOOK WAS TRULY A LABOR OF LOVE during the many days and late nights spent putting it together. Time flew, and before I knew it, the days had turned into weeks, and the weeks had turned into months. The end result is something that I am really proud of.

None of this would have been possible without the support of everyone at Wiley, who I'd like to give a big thank you! Thank you all for the time and effort spent making things happen.

I'd like to give special thanks to Scott Meyers, Acquisitions Editor, and Kenyon Brown, Project Editor, for driving the project. Thank you for the belief, guidance, and words of encouragement that were given throughout. Also, thank you to Darren Osadchuk, Technical Editor, for his technical advice and all important feedback.

I'd also like to thank the development team at Sony Ericsson — Velimir Karadzic, Enrique Garcia, and Youness Ghanim — who gave some of their time to review and give feedback.

Finally, many thanks to all my family and friends, for the love and support they gave me during the time of writing. I couldn't have done it without them, especially my darling wife, Joanna.

—Jermaine G. Anderson

www.it-ebooks.info

CONTENTS

INTRODUCTION *xvii*

CHAPTER 1: FLASH IS MOBILE

Mobile Devices **1**
The Mobile Ecosystem **2**
Participants 2
Recommended Reading 4
Engaging Experiences That Work on Small Screens **4**
What Is Flash Lite? 5
Past, Present, and Future 5
The Open Screen Project 6
Multiple Devices and Platforms 7
OEMs Supporting Flash Lite Player Development 7
Types of Flash Lite Mobile Content 7
Flash Lite Architecture **8**
Flash Lite Player 3.x Features 9
Summary **12**

CHAPTER 2: GETTING STARTED

Developing for Flash Devices **13**
What You Will Need 14
Using Device Central **15**
What Are Device Sets? 15
Using the Online and Local Libraries 16
Understanding Device Profiles 16
Creating Device Sets 17
Comparing Device Profile Information 18
Creating New Flash Mobile Documents 21
Creating a "Hello World" Example **23**
Using the Emulator in Device Central **26**
Memory and Device CPU 27
Testing on Mobile Devices **30**
Devices with Flash Lite Pre-Installed 31
Remote Devices 31
Summary **31**

www.it-ebooks.info

CHAPTER 3: OBJECT-ORIENTED PROGRAMMING FOR FLASH LITE DEVELOPMENT

AS2 Revisited **33**
Object-Oriented Programming for Flash Lite **35**
Why OOP? 35
An OOP Example for Mobile 35
Mobile Device Considerations **47**
Using fscommands to Communicate with the Device 47
Using Device Capabilities **60**
System.capabilities 61
Events and Event Listeners **64**
Using the EventDispatcher Class 64
Keys, Buttons, and Touch Input **68**
Using the Key Class 68
Using the ExtendedKey Class 71
Handling Touch Interaction 72
Summary **74**

CHAPTER 4: UI COMPONENTS

Sony Ericsson and Forum Nokia Component Libraries **76**
Using Text **77**
Using Buttons **77**
UI Component States 78
Using the Component Inspector 78
Using Soft Keys **82**
Using Status Bar **83**
Using Title **84**
Using Check Boxes **84**
Using Radio Buttons **87**
Using Lists **90**
Creating a Single Row List 90
Using Modal Dialogues **96**
Using Visual Indicators **98**
The Progress Indicator 98
The Wait Indicator 100
Using Sliders **100**
Using Scrollable Areas **102**
Using the Scrollable Area Component 102
Using the Scrollable Text Component 103
Using Notification **104**
Summary **105**

www.it-ebooks.info

CHAPTER 5: PUREMVC ACTIONSCRIPT 2.0 FRAMEWORK

Design Patterns and Development Frameworks **107**
Time for a Development Framework 107
Design Patterns Used in PureMVC 109
PureMVC Key Concepts Explained **111**
The Facade 111
Notifications and the Observer 114
Commands and the Controller 116
Mediators and The View 119
Proxies and the Model 125
Building a PureMVC Example **126**
Creating a Model for the Example app 128
Creating the Example App's Controller 132
Exploring the View 134
Viewing the Application in Device Central 136
Summary **136**

CHAPTER 6: CREATING A TV LISTINGS APPLICATION

The Concept **139**
TV Genius Search Engine API **140**
Mandatory Search Parameters 140
Optional Search Parameters 140
Response Format for an API Call 141
Functional Requirements **142**
Building the Application **142**
The .fla File 143
Defining ApplicationFacade 143
Creating a Model for the TV Listings Application 144
Using XML Data 149
Creating the TV Listings Application's Controller 158
Exploring the View 165
Viewing the Application in Device Central 174
Security Considerations: Loading Data **175**
The crossdomain.xml File 176
Setting the Sandbox Type 176
Setting the Playback Security 176
Summary **177**

www.it-ebooks.info

CHAPTER 7: CREATING A MEDIA CONSOLE

The Core Media Classes **179**
Using NetConnection 179
Using NetStream 180
Using the Video Object 184
Using the Sound Object 185
Streaming Audio and Video **190**
Streaming FLV Video 190
Streaming MP3 Audio 191
System.capabilities Revisited 192
Building the Application **192**
The .fla File 192
Defining ApplicationFacade 193
Creating a Model for the Media Console 196
Creating the Media Console's Controller 218
Exploring the View's Mediators 224
Viewing the Application in Device Central 238
Summary **239**

CHAPTER 8: CREATING AN IMAGE VIEWER CLIENT

The Challenges **241**
Supported Images 242
Loading Multiple Images 242
Memory Considerations 243
Handling Image Sizes 243
Components Used in the Image Viewer **243**
MovieClipLoader 244
The SharedObject Class 254
Building the Application **255**
The .fla File 256
Defining ApplicationFacade 256
Creating a Model for the Image Viewer 258
Exploring the View 268
Creating the Image Viewer's Controller 275
Viewing the Application in Device Central 283
Summary **283**

CHAPTER 9: CREATING A TWITTER CLIENT

Exploring the Twitter API **286**
Twitter Fundamentals 286
Twitter API Methods 288

www.it-ebooks.info

The SWX Format **292**
What Is SWX? 292
User Timeline 294
Building the Application **298**
The .fla File 299
Defining ApplicationFacade 299
Creating a Model for the Twitter Client 300
Creating the Twitter Client's Controller 316
Exploring the View 322
Viewing the Application in Device Central 336
Summary **336**

CHAPTER 10: USING NOKIA'S S60 PLATFORM SERVICES API

S60 Platform Services Overview **337**
Download and Install the Library 338
How to Use the API Methods 338
Using AppManager **343**
API Features 343
Using Calendar **347**
API Features 347
Using Contacts **353**
API Features 353
Using Landmarks **360**
API Features 360
Using Location **365**
API Features 365
Using Messaging **369**
API Features 370
Using Media Management **378**
API Features 378
Other S60 Platform Services **383**
ActionScript Sensor Service API 383
ActionScript Logging Service API 384
ActionScript SystemInfo Service API 384
Summary **384**

CHAPTER 11: CREATING A WEATHER CLIENT

Google APIs **385**
Using Google Weather API 385
Weather Conditions 388

www.it-ebooks.info

Forecast_Information 388
Using Google Map API 390
Wireframes and Design 397
Building the Application 398
The .fla File 399
Defining ApplicationFacade 399
Creating a Model for the Weather Client 400
Creating the Weather Client's Controller 421
Exploring the View 426
Viewing the Application in Device Central 436
Summary 437

CHAPTER 12: USING SONY ERICSSON'S PROJECT CAPUCHIN PLATFORM SERVICES API

Project Capuchin Platform Services Overview 440
Project Capuchin API Services 440
How to Use the API methods 441
Using Accelerometer 441
API Features 441
Testing the API in Device Central 446
Using Bluetooth 448
API Features 448
Using Calendar 454
API Features 454
Using Contacts 457
API Features 458
Using File 461
API Features 461
Using I18n 465
API Features 465
Using Messaging 470
API Features 470
Using Persistency 471
API Features 471
Using Radio 475
API Features 475
Other Project Capuchin Services 483
Location 483
Multimedia 483
Summary 484

www.it-ebooks.info

CHAPTER 13: PACKAGING FLASH LITE APPLICATIONS FOR DISTRIBUTION

Packaging Content **485**
Packaging for Different Platforms 486
Using Swf2Jar **486**
Creating a .jar File 487
Creating a .sis File **489**
Obtaining a Unique Identifier (UID) 489
Creating the Package (PKG) File 491
Using makesis.exe 496
Application Signing **496**
Certificates 497
Untrusted Content 497
Using makekeys.sis 498
Using signsis.exe 499
Signing for Windows Mobile 500
Signing for Symbian S60 500
Signing for Java 500
Signing Programs 501
Publishing to Aggregators 501
Application Icons **501**
Using mifconv.exe 502
Using epocrc.exe 503
Nokia Flash Packaging Tool **505**
Introducing Ovi from Nokia **506**
The Ovi Store 506
Introducing PlayNow Arena from Sony Ericsson **507**
Sony Ericsson Content Submission 507
Summary **508**

APPENDIX: FLASH LITE 3.X ACTIONSCRIPT 2.0 QUICK REFERENCE: FROM ARRAY TO XMLSOCKET **509**

INDEX ***537***

www.it-ebooks.info

INTRODUCTION

WITH THE RECENT EXPLOSION OF MOBILE APPLICATIONS and app stores, there has never been a better time for developers to begin creating content for Flash-enabled mobile devices. With many of the leading device manufacturers supporting Flash Lite, there are lots of opportunities for existing Flash developers to get their content into the hands of mobile consumers. The wide consumer reach of Flash-enabled devices is what also makes Flash Lite a very attractive route for both new and experienced application developers to begin developing mobile content.

Professional Flash Lite Mobile Development covers a variety of advanced topics for developing Flash mobile content, with the core focus on developing applications.

This book gives extensive code examples detailing how to create several Flash Lite mobile applications from the ground up utilizing Web service APIs, images, video, and audio. The book has great appeal for anybody wanting to monetize and showcase their mobile content on a global scale.

WHO SHOULD USE THIS BOOK

Programmers and developers of all experiences will be able to use the book as a reference on how to build content for mobiles and devices using Adobe Flash Lite, from concept to completion.

This book is aimed primarily at mobile developers looking to create and distribute new Flash mobile applications using Flash Lite.

This book is also designed to help newcomers and experienced mobile developers in Flash and other platforms get the most out of Flash Lite, and also provide those looking to create mobile applications with a solid base and framework concepts, which they can apply to future iterations of the Flash technology.

WHAT THIS BOOK COVERS

Professional Flash Lite Mobile Development introduces advanced topics, covering the key aspects of Flash Lite mobile development:

- It guides the reader through a series of code samples and example applications that explore cool concepts, best practices, and the core features of the Flash Lite player.
- It introduces the PureMVC framework as the basis for structuring the core example applications, familiarizing the reader with well-structured object-oriented programming with ActionScript.

www.it-ebooks.info

- It goes beyond the boundaries of the Flash Lite player, exploring the platform APIs of two of the leading mobile device manufacturers supporting Flash Lite and Flash-enabled devices.
- It provides details and best practices on how to package and distribute Flash Lite applications.

Chapter-by-Chapter Description

A description of each chapter follows:

- **Chapter 1 — Flash Is Mobile**: This chapter provides a brief discussion of mobile devices and Flash, as well as its role in the mobile ecosystem, and a look at Flash Lite and what it is. It also takes a retrospective look at the iterations of Flash Lite and then covers the core features of the technology that the reader explores throughout the book.
- **Chapter 2 — Getting Started**: This chapter focuses on getting started with Flash Lite mobile development, taking a look at the two essential tools used in developing and testing Flash mobile and device content: Adobe Flash CS4 Professional and Adobe Device Central CS4.
- **Chapter 3 — Object-Oriented Programming for Flash Lite Development**: This chapter includes an overview of ActionScript 2.0, and common object-oriented programming concepts that can be used in coding Flash Lite applications. It also covers language elements specific to Flash Lite mobile development.
- **Chapter 4 — UI Components**: This chapter explores the excellent Sony Ericsson UI component library used for developing Flash Lite user interfaces. Many of these components are used throughout the book.
- **Chapter 5 — PureMVC ActionScript 2.0 Framework**: This chapter provides an in-depth look at PureMVC and the several development concepts and frameworks that can be used to structure code for Flash Lite applications.
- **Chapter 6 — Creating a TV Listings Application**: This chapter takes a detailed and hands-on approach to developing a TV listings application.
- **Chapter 7 — Creating a Media Console**: This chapter provides detailed insight into, and approaches to, developing rich audio and video Flash Lite applications.
- **Chapter 8 — Creating an Image Viewer**: This chapter covers aspects of using images in Flash Lite and provides a detailed walkthrough of an image application.
- **Chapter 9 — Creating a Twitter Client**: This chapter introduces the Twitter API and SWX format, and then explores the code behind a Flash Lite Twitter application from the ground up.
- **Chapter 10 — Using Nokia's S60 Platform Services API**: Takes an in-depth look at extending Flash Lite mobile applications beyond the features of the Flash Lite player, using the S60 Platform Services API from Nokia.
- **Chapter 11 — Creating a Weather Client**: Following on from the S60 Platform Services API, this covers design and development of a Weather Client, a location-based service application that uses a combination of the mobile device hardware, and the Google Maps and Google Weather APIs, to retrieve the latest weather forecast.

www.it-ebooks.info

- **Chapter 12 — Using Sony Ericsson's Project Capuchin Platform Services API**: Takes an in-depth look at using the Project Capuchin Platform Services API from Sony Ericsson, allowing the reader to extend Flash Lite mobile applications beyond the features of the Flash Lite player.
- **Chapter 13 — Packaging Flash Lite Applications for Distribution**: Covers how to package completed Flash Lite mobile applications for consumers, introduces aggregators, and the Sony Ericsson and Nokia content distribution channels.
- **Appendix — Flash Lite 3.x ActionScript 2.0 Quick Reference: From Array to XMLSocket**: Provides a handy reference for all the class methods, properties, operators, and events that are found in ActionScript.

HOW THIS BOOK IS STRUCTURED

While each chapter of the book follows on from one to the other, the book is written in such a way that it allows the reader to pick up and start from any chapter.

I recommend reading Chapter 5 before tackling the code examples, as Chapter 5 provides the background for the code examples in Chapter 6 through to 9, and Chapter 12.

Every other chapter in the book can be read in isolation.

WHAT YOU NEED TO USE THIS BOOK

You will need to have one of the following operating systems:

- Windows, Mac or Linux

To use the code samples and run the example applications in this book you will need the following:

- Adobe Flash CS4 Professional
- Adobe Device Central CS4
- Adobe Extensions Manager

You do not explicitly need a Flash Lite-enabled mobile device to complete the applications in the book.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.



Boxes with a warning icon like this one hold important, not-to-be forgotten information that is directly relevant to the surrounding text.

www.it-ebooks.info



The pencil icon indicates notes, tips, hints, tricks, or asides to the current discussion.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show file names, URLs, and code within the text like so: `persistence.properties`.
- We present code in two different ways:

```
We use a monofont type with no highlighting for most code examples.
We use bold to emphasize code that's particularly important in the present
context or to show changes from a previous code snippet.
```

The example code block gives two lines of code, the one with no highlighting is general code, used in the majority of examples in the book. Whereas the second line has grey highlighting to emphasize the code is important in the present context. Here I'd want you to focus on the text color property `_textColor` property of the object called `statusTitle`.

SOURCE CODE

The days when code was small and easily explained in a few written pages are gone. Today, many Flash applications are thousands of lines of code with large supporting libraries. Due to the size of many 3D applications, you no longer have the luxury of showing every line of code that goes behind each application. And in many instances, in this book, after a discussion of how code works the reader is sent to the book's website to download the entire application for review.

The amount of code written for the book was significant (about 119,000 files) and as the book was written the versions of PV3D and other supporting software changed. So as opposed to trying to keep track of every different version of PV3D, the version of PV3D (and other supporting software) used to create each application was included with each application.

All of the source code used in this book is available for download at `http://www.wrox.com`. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists), and click the Download Code link on the book's detail page to obtain all the source code for the book.



Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-54748-9.

Once you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at `http://www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

www.it-ebooks.info

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save other readers hours of frustration and at the same time you will be helping us provide even higher-quality information.

To find the errata page for this book, go to `http://www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at `http://www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `http://www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at `http://p2p.wrox.com`. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com` you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `http://p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.



You can read messages in the forums without joining P2P but in order to post your own messages, you must join.

www.it-ebooks.info

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

www.it-ebooks.info

1

Flash Is Mobile

WHAT'S IN THIS CHAPTER?

- Developing for mobile devices and small screens
- Participants in the mobile ecosystem
- Evolution of the Flash Player optimized for mobile devices
- Core features and architecture of the Flash Lite Player

As a title for the first chapter in the book I thought I'd pay a little homage to one of the early encounters I had with Flash on mobile devices. The title is actually adopted from the Macromedia marketing header during the launch of Flash Lite 2.0 at Macromedia MAX 2005; there have been other noticeable catchy slogans for Adobe's mobile and device marketing since then, such as "Engage with Flash," but here we have a strap line that hits the nail on the head and sets us on our way . . . "*Flash Is Mobile*. . ."

Flash has actually been *mobile* for a number of years now. In order to understand in part how Flash can be utilized on mobile devices, *how* and *where* you, the developer, can leverage the technology, in this chapter you'll take a brief look at the short history and features of Flash Lite Player, the predominant Flash Player runtime used in mobile devices.

MOBILE DEVICES

Since the turn of the century, mobile phones have come a long way from functioning as simple communications devices and today these devices are providing us with everything we need for our daily functions.

Today, mobile devices allow us to listen to our favorite songs, watch recorded video clips and feature films, and record or watch the television shows we have missed. They come with cameras so we can take digital photos and share them with others. We can also use mobiles

www.it-ebooks.info

to play games and, right now, people all over the world are using their mobile devices to send messages, access the Internet, send and receive email, and use various services daily.

> *"The mobile phone is the first digital device most people look at in the morning, and the last device before going to sleep at night. It is the only digital device many take to the bathroom, and the only one we carry with us all day."*
>
> —Tomi T. Ahonen and Alan Moore, *"Communities Dominate Brands"*

The possibilities for the mobile device and its *applications*, right now and beyond, are endless and it is developing mobile applications that is the core focus of this book.

As the underlying technologies used in mobile device handsets have improved, so have the capabilities. In today's global mobile handset market, consumers have a wide variety of mobile phones to choose from, each with a diverse range of features, aiming to cater for our desires and to make our lives that little bit better.

There are two commonly used terms that categorize the types of mobile phones on the market, the "feature phones" and the "smartphones." The feature phone is a mobile device with limited or restricted capabilities, where the underlying hardware technology generally tends to be less expensive and at the lower end of the market. Smartphones, on the other hand, are mobile devices with advanced features and capabilities similar to PCs, such as Internet access, and it is this type of mobile device that is becoming more and more popular in the handset market.

For the Flash Lite developer, there is a range of devices that you can target for your applications, and in this book you will focus on developing mobile applications for the smartphone. You'll cover a wide range of topics relating to Flash Lite mobile application development, through developing a TV Listings Guide, a media Console, an Image Viewer, a Twitter Client, and a Weather Client. All are examples of content that can be consumed by the mobile end user.

Consuming information via a smartphone, anytime, anywhere has never been easier since the introduction of high-speed data networks such as 3G, but some devices are also capable of accessing Wi-Fi networks. But there are several key players involved in ensuring that services and information are delivered to our devices, in a highly competitive, but also thriving, mobile market.

THE MOBILE ECOSYSTEM

Working in the mobile space has its challenges, not just from a developer's code and development point of view but also for other key participants involved in the mobile ecosystem.

Participants

Figure 1-1 shows the key participants of the mobile ecosystem adapted from Adobe.

The following lists the mobile ecosystem's key participants, as shown in Figure 1.1; these will be discussed shortly:

www.it-ebooks.info

- Original Equipment Manufacturers (OEMs)
- Network operators
- Media owners
- Content producers
- Aggregators
- App stores

For each of the key participants mentioned here in the mobile ecosystem, you'll notice that there is a great deal of overlap in goals, and a need for mobile applications to succeed. Flash offers the distinction from other mobile technologies in that it is capable of running on different platforms.

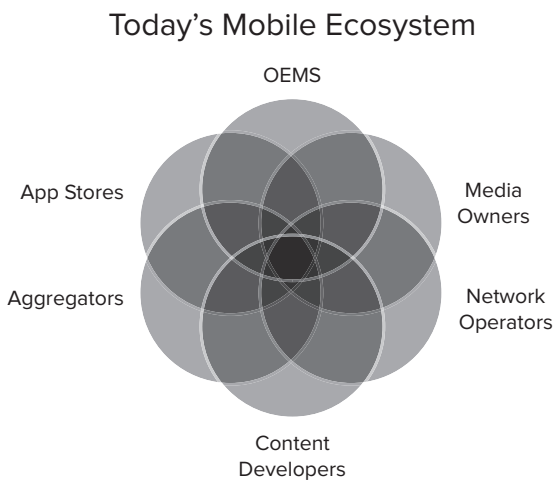


FIGURE 1-1

OEMs

Original equipment manufacturers (OEMs) manufacture the mobile handsets. They are continually developing new products for the market. They are the innovators responsible for fusing the latest hardware with the software platform that each handset model runs on. The OEMs determine the operating system (OS) used by the device, which can also determine the device's features and capabilities. OEMs work directly with network operators to ensure that networking hardware is also integrated. For OEMs, it's imperative to have software on the device that can engage the end user and, essentially, distinguish their products from others in the market.

Network Operators

Network operators manage the network adopted by the consumer. The operators provide the services and channels of content that are accessed by their subscribers. Operators are also known as *carriers*. They are always looking for ways to improve services to bring more subscribers to their networks.

Media Owners

Media owners effectively own the rights to distribute particular content. Media owners want to be able to distribute their brands to as many domains as possible. Like the OEMs, media owners want to distinguish their brands and ensure that they stand out from the crowd.

Content Producers

Content producers represent the designers and developers who produce mobile content. They often work with the brand guidelines from the media owners to design, develop, and produce mobile content.

Aggregators

Aggregators syndicate and distribute mobile content through online portals and also to the network operators ready for consumption by the consumer. Aggregators were early adopters of syndicating

www.it-ebooks.info

mobile content, who saw great potential of collecting mobile content developed by third-party content producers and distributing it.

App Stores

Unless you've been living in a cave for the past year, you know that app stores are where consumers can find content to install on their mobile phones. With devices such as the iPhone, apps, applications, and app stores have been a huge commercial success.

As a Flash mobile developer, the opportunities to generate engaging experiences on mobile phones are vast. Innovative Internet services and applications that require data services are at the heart of the mobile ecosystem and are two of the driving forces behind mobile usage.

Recommended Reading

For a detailed breakdown of the mobile ecosystem, I recommend that you read the Strategy Analytics PDF "*Understanding the Mobile Ecosystem*" .pdf file found on the Adobe Web site at `http://www.adobe.com/devnet/devices/articles/mobile_ecosystem.pdf`.

ENGAGING EXPERIENCES THAT WORK ON SMALL SCREENS

Flash has changed the way that people consume and engage with content on the Internet around the world. From animation to video to rich media platforms, today Flash continues to push the boundaries for our digital world and is establishing itself as the platform for the next generation of publishing across multiple screens.

Developing for mobile devices using Flash requires a slightly different train of thought when compared to desktop or Internet browsers:

- **Screen size** — Developing for smaller screens will mean you have to think more *inside of the box* as well as outside it; in particular, you will need to pay more attention to the detail provided in your applications.
- **Processing power** — With most smartphones having a lower or limited CPU power compared to home PCs, performance measurements need to be taken into consideration.
- **Interaction and navigation** — There is no mouse on a mobile device. The majority of smartphones have keypads; some have touch screens; others have both, which means that, during development, you have to really carefully consider *how* the end user interacts with the Flash application on the device.
- **Memory** — Ultimately mobile devices tend to have less available memory than desktop computers. Regardless of what Flash applications you develop, they will have an impact on memory, and you will need to be aware of the memory consumed by your applications and the limits to which you can go.

If you are already a mobile developer, then no doubt these issues are already familiar to you, and you know that developing for mobile is not the same as developing for the desktop or Web browsers on personal computers.

www.it-ebooks.info

What is Flash Lite?

Adobe describes Flash Lite as an "optimized version" of the Flash Player software that is used in PC's for desktop and Web browsers. The optimized runtime particularly addresses aspects of content running on devices with limited capabilities.

The .swf (SWF) file extension is synonymous with Flash and all versions of the Flash Player, including Flash Lite. The SWF format was introduced in 1998, and the Flash Player is the runtime engine that executes bytecode contained within the .swf files. You'll be making many a SWF throughout the course of this book.

At the time of writing, there are over 700 types of devices supporting the Flash Player. Flash Lite has been installed on more than one billion devices and can be directly distributed to millions of open OS smartphones.

So what's the story on Flash Lite?

Past, Present, and Future

In what now seems like *"a long time ago in a galaxy far, far away . . . "*, the software company Macromedia (acquired by Adobe) saw the potential of content-rich media on mobile devices. With its early release of Flash Lite in Korea, it achieved its goal of replicating Flash content, produced on personal computers, on other devices, albeit with limited features compared to those on the browser- and desktop-based Flash Player.

The browser-based versions of the Flash Player have always been out of sync with the mobile version, because of the limitations of the device hardware.

Flash Lite 1.*x*

Flash had already been introduced to Pocket PC devices by the time Flash Lite first appeared as Flash Lite 1.0 on the Japanese NTT DoCoMo i-mode system, which offered mobile Web sites and content such as animations, games, wallpapers, and screensavers for consumers to download over their network. Flash Lite 1.0 was then superseded by Flash Lite 1.1, a release that had a number of enhancements, including network access and integration with the handset.

Flash Lite 1.1, allowed existing Flash developers, who created similar content for Web sites, to leverage their skills. After a few years of progress with Flash on mobile in the Asia Pacific (APAC) region, the Flash Lite 1.1 player made its way to developers in other regions of the world: Europe, the Middle East, and Africa (EMEA); North America, and South America. This iteration of the Flash Lite Player signaled the start of a new dawn for Flash on mobile devices, and as the developer community became more established, licensing agreements were made with OEMs to pre-install the Flash Lite Player on mobile handsets allowing consumers to view compatible content.

Flash Lite 2.*x*

A few years after the release of 1.1, Flash Lite 2.0 was introduced with performance improvements and new features, including the ability to play both device video and sound, attracting further

www.it-ebooks.info

interest from OEMs and establishing the player as a fully rich media player for the mobile market among its competitors.

Developers from other mobile software development backgrounds, attracted by the ease of using and rapid development possible with 2.0, also found Flash Lite to be a great interactive mobile solution.

The open source application development platform established by Qualcomm BREW (Binary Runtime Environment for Wireless) later was given a slightly different implementation of Flash Lite, in Flash Lite 2.1.

Flash Lite 3.*x*

Not long ago Flash 3.0 opened up new avenues for Flash on mobile. Fueled in part by a "revolution" in Flash video on the Internet, the more evolved features in version 3.0 meant that the player would allow playback of video directly into the Flash Player, another boost for the multimedia capabilities of the player.

Other features of the player included performance enhancements and the ability to stream media through Adobe Flash Media Server (FMS). More recently Flash Lite 3.1.*x* became the first iteration of the Flash Lite Player that can be distributed with content created by developers.

This book primarily covers Flash Lite 3.*x*. There is another version of the Flash Lite Player currently in development that will be backwardly compatible; however, at the time of writing, this is not available to developers.

The Open Screen Project

The Open Screen Project (`http://www.openscreenproject.org/`) is an industry-wide collaboration to establish a "consistent runtime environment" for experiences across a variety of screens and devices. The initiative recognizes the Adobe Flash platform as the key technology to deliver consistency and provide consumers with the rich experiences they demand, whether in a browser or on the desktop, a mobile device, TV, or a gaming console. This is an ambitious project, one that remains central to Adobe, which leads the initiative, and one that will also be central to the strategies of the majority of content owners in the not too distant future.

Flash Player 10.1

At the time of writing, the next increment of the Adobe Flash Player runtime, Flash Player 10.1, is going through its public beta for developers. This is the first runtime release adopted by the Open Screen Project, which enables Flash content to run across numerous platforms and devices, including mobile. This is the first full Flash player targeted at both mobile devices and PCs; the announcement of the release signals that mobile devices are becoming more powerful, capable of presenting content that matches that of high-end desktop technology.

Flash Lite 3.*x* content will be supported in both Flash Player 10.1 and the next "unreleased" version of the Flash Lite player, and it is said that devices containing this are soon to be referred to simply as "Flash enabled."

www.it-ebooks.info

Multiple Devices and Platforms

Although the sole focus of this book is on developing mobile applications, you should be aware the Flash Lite Player 3.*x* has been ported to a variety of consumer electronics and numerous platforms, including:

- Nokia
- Sony Ericsson
- HTC
- Microsoft
- Chumby
- iRiver

At the same time that the adoption of Flash Player 10.1 will undoubtedly grow in 2010, the integration of the Flash Lite Player into devices will also continue to grow for a while and is estimated to peak in the billions by the end of 2010.

OEMs Supporting Flash Lite Player Development

Device manufacturers Nokia and Sony Ericsson are among a number of OEMs supporting Flash mobile development, and these two in particular have resources that you can utilize in your Flash development. One of the very early debates among developers during the early release of Flash Lite was whether Flash Lite or Java Platform, Micro Edition (JME) would prevail as the mobile technology of choice. They coexist, and Sony Ericsson has enabled developers to leverage the benefits of both technologies in the Project Capuchin initiative, which has enabled the Flash Player to be bundled in a J2ME wrapper to provide the additional features of a device; this also provides a good crossover between the two software entities. Extending Flash Lite applications with Project Capuchin is covered in Chapter 12.

Types of Flash Lite Mobile Content

One of the challenging aspects of mobile application development is the complexity of devices, operating systems, and capabilities of each device. The key advantage of Flash is that it can run on multiple handsets and operating systems. However, not all content types will support all Flash Lite features; even if the Flash Lite version is the same, there still could be some aspect of Flash Lite that isn't supported.

Flash Lite content comes under particular categories, including:

- Screensavers
- Ringtones
- Standalone applications

This book contains content specifically created for the *standalone* versions of Flash Lite.

www.it-ebooks.info

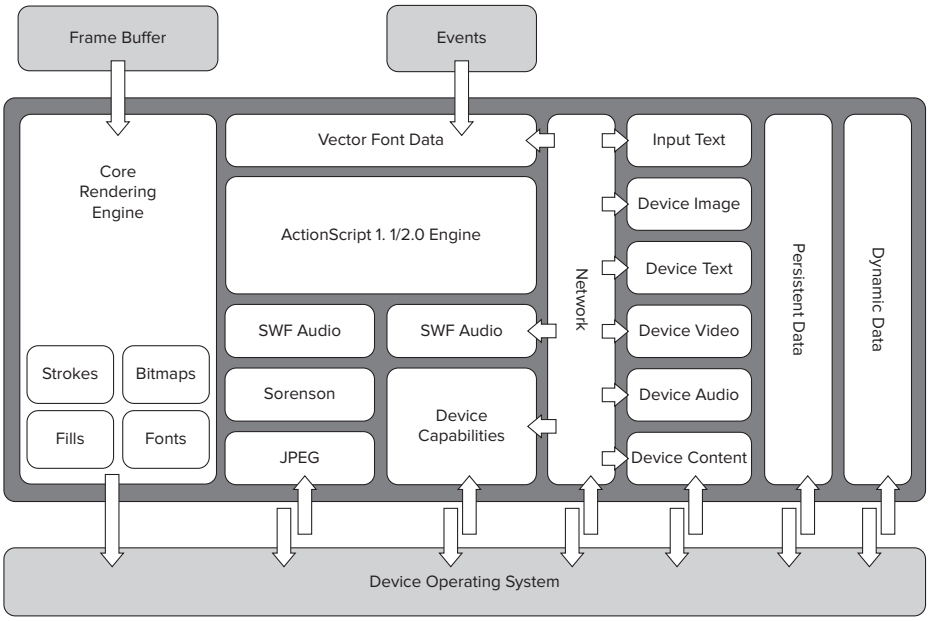
Adobe Device Central provides a more comprehensive list of content types that you can create using Flash Lite. It also provides the full set of properties and features supported by various mobile device handsets. You'll learn more about Device Central in Chapter 2.

The aim of each application covered in the book is to provide you with practical solutions for developing a wide range of Flash applications for mobiles devices.

FLASH LITE ARCHITECTURE

In this section, we'll take a look at the Flash Lite 3.*x* Player architecture and go over the features that you'll be covering in the next section.

In Figure 1-2, you can see the overall architecture of the Flash Lite 3.0 player.



SOURCE: HTTP://WWW.ADOBE.COM/PRODUCTS/FLASHLITE/ARCHITECTURE/

FIGURE 1-2

The main aspects of the Flash Lite runtime can be seen as:

- Core Rendering Engine
- ActionScript 1.1/2.0 Engine
- Vector Font Data
- SWF Audio, a component enabling audio to play back
- Sorenson

www.it-ebooks.info

- JPEG
- Device Capabilities
- Device Content, Audio, Video, Text, and Image
- Persistent Data
- Dynamic Data

You'll cover aspects of the architecture in the next section.

Flash Lite Player 3.x Features

The majority of the Flash Lite Player features shown in the architecture diagram are covered throughout the chapters of this book:

- ActionScript 2.0
- Device features and capabilities
- Loading external resources
- Video support
- Audio support
- Font and text support
- Images
- Persistent data
- Dynamic data and XML support

Flash Lite gives the Flash mobile platform a lot of features, such as ActionScript 2.0 for accessing and controlling device keys and functions, networks and connectivity, audio playback, event and streaming sound, different types of texts, fonts, navigation controls, and images.

ActionScript 2.0

Although ActionScript 3.0 (AS3) has been around for a while now, Flash Lite 3.*x* fully supports the ActionScript 2.0 (AS2) engine. Based on the ECMA 262 standard, AS2 is a modern programming language that supports dot-syntax and object-oriented development. This book fully covers examples written in AS2.

You'll cover AS2 development in Chapter 3.



At the time of writing this book, the beta for the Flash Player 10.1 using version AS3 was announced. There is currently no developer edition of Flash Lite that supports AS3.

www.it-ebooks.info

Device Features and Capabilities

Every mobile device handset has its own particular features and capabilities, implemented by the OEMs. Video playback, sound support, and predictive user input text are all examples of features that may or may not be supported across a different range of mobile devices.

With the Flash Lite 3.*x* Player, you have ways of determining specifically which features or capabilities are supported by the device dynamically through AS2 properties and functions:

- `System.capabilities`
- `fscommand()` and `fscommand2()`

Both these functions allow you to target aspects of your applications during development. By using `System.capabilities`, you can access the properties of the device, such as determining the language. By using *fscommands* you can access and adjust device features directly from your mobile applications, such as monitoring the battery and network signal levels, sending text messages, making phone calls, altering the device's backlight, and setting the device to vibrate.

You'll learn more about accessing device features and capabilities in Chapter 3.

OEMs Nokia and Sony Ericsson also have APIs specifically for developing Flash Lite applications for their supported handsets, which allow you to access more advanced features of their devices and extend your Flash Lite applications. Features such as sending files via Bluetooth, gaining access to the Accelerometer, GPS or the device's file systems are examples of features which need to be retrieved by the Flash Lite Player.

You'll learn more about extending Flash Lite applications in Chapters 10 and 11.

Loading External Resources

Network connectivity enables Flash Lite 3.*x* applications to dynamically load data and resources, that reside externally, into the main SWF application. Multimedia, text-based content, and data objects are all resources supported externally from folders residing on the actual device or from a Web server over HTTP.

You'll learn more about loading external resources across the chapters in this book.

Audio Support

Flash Lite 3.*x* supports a range of file formats for audio playback, including MIDI, MP3, Pulse Code Modulation (PCM), Adaptive Differential PCM (ADPCM), and SMAF.

You can embed sounds directly into the Flash Lite application or load them into the application over HTTP or from a location on the device. Streaming audio via RTMP using Adobe Flash Media Server (FMS) is also supported.

You'll learn all about using sound in your applications in Chapter 7.

Video Support

Flash Video (FLV) is widely regarded as the *"most popular video format"* on the Internet. Support for FLV via the On2 VP6 and Sorenson video codecs, was one of the key features introduced in

www.it-ebooks.info

Flash Lite 3.0 allowing for the format to be played directly within the Flash Lite Player. The Flash Lite Player for mobile now includes support for H.264, the standard used in high definition (HD) televisions, game consoles, and video players.

Flash Lite also supports video playback of other video formats supported by the majority of mobile devices, including 3GPP and MPEG-4 files.

Video files can be embedded directly into the Flash Lite application, and they can also be loaded into the application externally over a HTTP server or from a location on the device itself at runtime. Delivery of video can also be made over the Real Time Messaging Protocol (RTMP) using media servers such as FMS and Red5.

You'll learn all about how to include video in your applications in Chapter 7.

Text and Font Support

Developing applications for Flash Lite allows you to include a variety of text components in your applications, including dynamic text fields, text for user input, and static text. Static fields are created during authoring time, whereas dynamic text and input text can be created during authoring time and at runtime.

You also have a choice between using device fonts or embedded fonts. With each choice there is a tradeoff; embedded fonts give you more design control of content but reduce the amount of memory available to the application. Using device fonts gives you less control over design but provide the reassurance that the content will render text.

Flash Lite 3.*x* supports the UTF-8 character set, including complex languages such as Chinese, Thai, Arabic, and Hebrew. For devices that support the features, Flash Lite also allows predictive text and inline text input. Text measurement and text wrapping are also supported by text fields and device-specific vector fonts for improved small text readability.



FIGURE 1-3

Navigation and Keys

The Flash Lite Player also accesses user input events from the device, including the keys for navigation up, down, left, right, and select, and also text, character symbols, and number keys from 0 to 9, *, and #, for example. Figure 1-3 shows an example of device keys supported on the Nokia E71 (from Device Central).

You'll cover navigation and interaction in Chapter 3.

Images

Using Flash Lite, you can render image files directly in your applications including PNG, JPEG, and GIF files. You'll cover including images in your applications in Chapter 8.

www.it-ebooks.info

Persistent Data

With Flash Lite you can also save data to the device and retrieve it for later use in your applications. You'll cover saving data in your applications in Chapter 8.

Dynamic Data and XML Support

Also available in Flash Lite 3.*x* is the ability to load and parse dynamic data, including native classes for loading and parsing of external XML. Several chapters throughout the book cover utilizing dynamic data and parsing XML in particular.

SUMMARY

In this chapter we wanted to give you an overview of the Flash Lite technology and provide a little background as to why the technology is used and where it is headed.

You took a look at the key participants in the mobile ecosystem and also gained insight into various aspects of the Flash Lite architecture, covering the core features of the Flash Lite 3.*x* Player. You've now learned what types of content are supported on devices with the Flash Lite Player installed, and at this stage you should have an idea of what you can develop with Flash Lite. In the upcoming chapters you will learn how to implement many of the features that the player supports, which were highlighted in this chapter.

The next chapter will take you through *getting started* with Flash Lite development.

www.it-ebooks.info

2

Getting Started

WHAT'S IN THIS CHAPTER?

- Using Adobe Flash CS4 and Device Central CS4 to create new Flash mobile projects
- Importing device sets into Device Central CS4
- Comparing features across multiple devices in Device Central CS4
- Creating a "Hello World" Flash Lite example
- Using the Device Central CS4 emulator
- Examining the core features of Device Central CS4 to aid in development and testing
- Understanding memory and CPU considerations for developing Flash Lite content
- Testing on devices

In this chapter, we'll take a look at the tools used in developing Flash Lite mobile applications, including the authoring environment Adobe Flash CS4 Professional and the device emulator and testing software Adobe Device Central CS4.

DEVELOPING FOR FLASH DEVICES

Given the variety of Flash Lite–supported devices now available, it shouldn't be surprising that not all of them have the same features. In the early days of developing Flash mobile applications, developers had no dedicated tools for previewing content, and they had to own a compatible device running the version of Flash Lite they needed. This made it very difficult for developers to know whether their content would run the way they intended on other devices.

www.it-ebooks.info

Things have certainly come a long way on that front, as the tools for developing Flash mobile content have become a lot more sophisticated. In this section we'll briefly cover the tools you'll use to create Flash mobile content, before focusing our attention on Device Central.

What You Will Need

Throughout this book you'll be following content, code listings, and code snippets developed with Adobe Flash CS4 Professional and Adobe Device Central CS4. Both of these software tools are supported on the following popular operating systems:

- Microsoft Windows (XP and Vista)
- Mac OS X

As long as you are running one of those operating systems, you should be able to install the following:

- Flash CS4 Professional
- Device Central CS4



The material in this book was created using Microsoft Windows Vista.

Adobe Flash CS4 Professional

You will use Flash CS4 to create the core assets of your mobile applications. You can also use Flash CS4 for coding the applications using ActionScript 2.x. Because this book focuses mostly on developing Flash Lite content using code and classes rather than using animation and the timeline, if you prefer you can use your own text editing software to create ActionScript classes. The supporting assets have been provided in later chapters.

Later in this chapter we'll cover the details of creating your first Flash mobile example using the Flash CS4 authoring environment.

Adobe Device Central CS4

Adobe Device Central CS4 provides you with a variety of features that aid in developing and testing *device*-based applications using Flash, including mobile.

The main features of Device Central are:

- It has a library of device profile information detailing various devices that run a version of the Flash Player.
- It has an emulator that provides developers with a way to simulate the experience of using an application on multiple devices.

www.it-ebooks.info

There are numerous advantages to using Device Central. You can view the properties of each device and the information maintained and updated by Adobe after testing device handsets submitted by OEMs.

The most significant aspect of this advantage is that it reduces time to market by speeding up the development process. You can also test your applications on multiple devices without the need to buy the handset that you are targeting.

In fact, the majority of the applications and example snippets provided across the chapters of this book can be tested in Device Central.



Adobe Flash CS4 Professional, which comes with Adobe Device Central, is, of course, a commercial product. You can get thirty-day trial versions that allow you to use all the features of the product, giving you just enough time to cover all the chapters in this book. To download a trial of Flash CS4, visit the Adobe Web site at `http://www.adobe.com/downloads/`.

USING DEVICE CENTRAL

When Device Central was introduced to the mobile developer community, it was received with plaudits for its innovation of features, simplicity, and ease of use. In this section we look at the features that make up the tool.

Here we'll cover the following aspects of Device Central:

- Device sets
- Online library
- Local library
- Device profiles
- New document
- Emulator

What Are Device Sets?

Creating applications for multiple devices is a lot easier when you have those devices in hand. But what happens if you don't have a particular device? One of the things you have to take into consideration when developing for Flash Lite in particular is *how* your application will perform on devices with different profiles. Device Central helps you target and focus development on a range of devices that you don't have, allowing you to address potential issues such as viewing content on devices with varying screen sizes and resolutions.

Device Central not only provides you with an interface for previewing applications, but it also allows you to compare the profiles of multiple devices in the Device Sets panel. The device sets

www.it-ebooks.info

are your own personal collection of device profiles that you can use to test and preview your content during your development. You can create multiple device sets in Device Central. Each set is effectively a folder that you can rename for a specific project, thereby making it easier to manage multiple mobile projects.

At the end of this section you'll create your first device set, and having done that will prepare you for the rest of the chapters in this book. To create a device set, you'll need to retrieve device profiles from the online and local libraries. Let's take a look at these aspects of Device Central next.

Using the Online and Local Libraries

The online library allows you to connect to a central repository containing the most up-to-date information on the majority of the devices that support Flash.

Device name, display size, color depth, screen resolution, and supported Flash Player version are all properties of a device profile, displayed in the online library section. The information retrieved from the online library can be refreshed manually by clicking the Refresh button in the panel's toolbar. While you're online, you can view and download the device profiles to your machine to review at any time, even offline if you wish.

The local library contains a collection of device profiles you have imported from the online library. You can add a device by dragging its profile from the online library or by right-clicking the profile and selecting Download to Local Library. By default, the local library has the Adobe Generic Phone, which is an example device profile used to give the maximum features offered by Flash Lite. There are a number of generic phones in the online library which you can download to test all the features of Flash Lite in the device. An example of a Generic Phone device template is shown in Figure 2-1.

The device profile listings in both local and online libraries can be grouped (filtered) by carrier, content type, display size, Flash version, manufacturer, or region. By default, device listings in these panels are not grouped by any category and have the filter set to *none*.



FIGURE 2-1

Understanding Device Profiles

When you highlight a device in either the online or local library panel, the Device Profiles panel displays detailed information about the device. This information is categorized into several subsections depending on the device selected.

Figure 2-2 shows the Nokia N95 8GB selected in the online library section of Device Central.

www.it-ebooks.info



FIGURE 2-2

With a device selected, you should be able to see the following subsections on the right side of the Device Profile panel:

- **General:** This section contains information about a device, such as its supported languages, regions in which it has been released, and supported networks and carriers. The section has core device information, including color depth, OS, platform, Flash version, and screen dimensions.
- **Flash:** This section details more information about the Flash Player integration on the device, such as memory, device integration, file access, text and font support, and user input types supported. Here you'll also find a list of features for image, sound, and video, covering various devices, and Flash Player–supported formats. The section also contains supported features for FLV and *fscommands*.
- **Bitmap:** This area details information about image content, including the supported image formats, background color, scaling, and alignment.
- **Video:** This section details information about video content for standalone content, including supported formats, background color, transformation, and alignment.
- **Web:** This area details information about the supported formats for the device's Web browser.

Now that we've covered aspects of the Device Profile panel, we'll take a look at creating a device set.

Creating Device Sets

In this section you'll run through an example of how to create a device set in Device Central. The first device you're going to add is the Nokia E71.

1. Open the Online Library panel and ensure that you are connected. Select Connect to view the online library of device profiles.

www.it-ebooks.info

After a short while a number of device profiles should start to populate the panel. During this short period you should see an icon indicating that this process is still in progress.

2. In the header of the Online Library panel are three buttons: The first is a grouping filter, the second is for search, and the third is a refresh button. Select the Manufacturer option from the grouping filter. Under the Nokia grouping, right-click on Nokia E71 and select Download to Local Library.
3. Ensure that the Device Sets panel is open by clicking the drop-down arrow.
4. Right-click Nokia E71 in the local library and then click New Device Set from Selection.
5. You will see that the Nokia E71 has been added in the Device Sets panel, under a folder named New Sets. Click the folder name to make it editable and then rename the folder Wrox Pro Flash Lite.

Once you've added a device to the Device Sets panel, you can use the device profile to create new Flash mobile projects. You will now also be able to preview Flash mobile content you've created using the Device Central Emulator by selecting this device profile from your new device set.

Comparing Device Profile Information

We'll now add a few more devices to our new device set.

1. Import Nokia N95 GB and Nokia 5800 XpressMusic into the local library, then drag them to the Wrox Pro Flash Lite folder.
2. Select the Wrox Pro Flash Lite device set folder. Notice that the Device Profiles panel shows a comparison charting the three devices.

Figure 2-3 shows a comparison of the three Nokia device profiles.

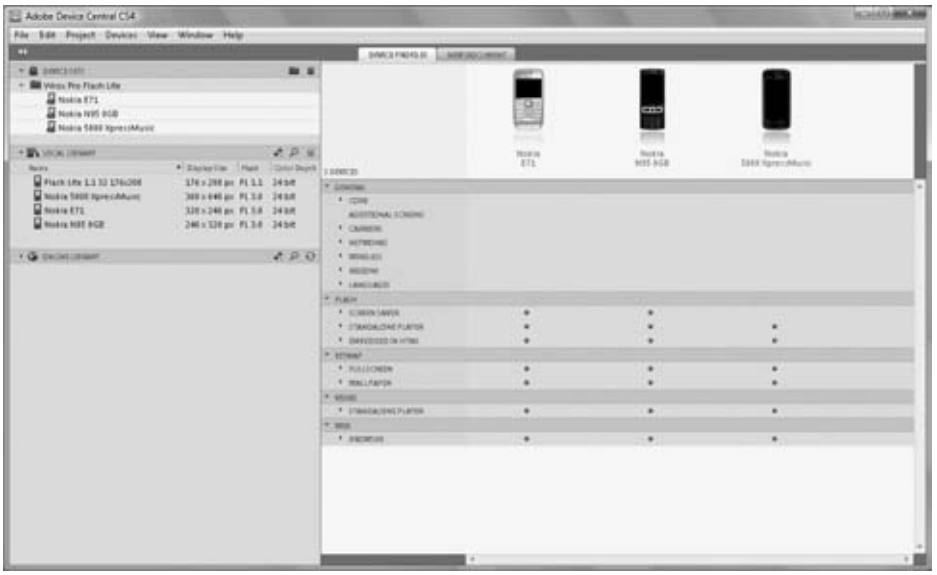


FIGURE 2-3

www.it-ebooks.info

In the comparison view the device profiles are arranged in columns. In each column you'll see the image of each device displayed above its manufacturer name and model name. Each device has its corresponding profile information below.

Next let's take a look at some of that information.

Display Sizes

We'll now take a look at comparing the display sizes of the device profiles, and demonstrate that comparing device profile information can affect display sizes.

You still should have the Wrox Pro Flash Lite folder (device set) highlighted, with the three devices showing in the Device Profiles panel. Next open General ➪ Core.

Figure 2-4 shows the Core information comparison for each of the three Nokia device profiles.

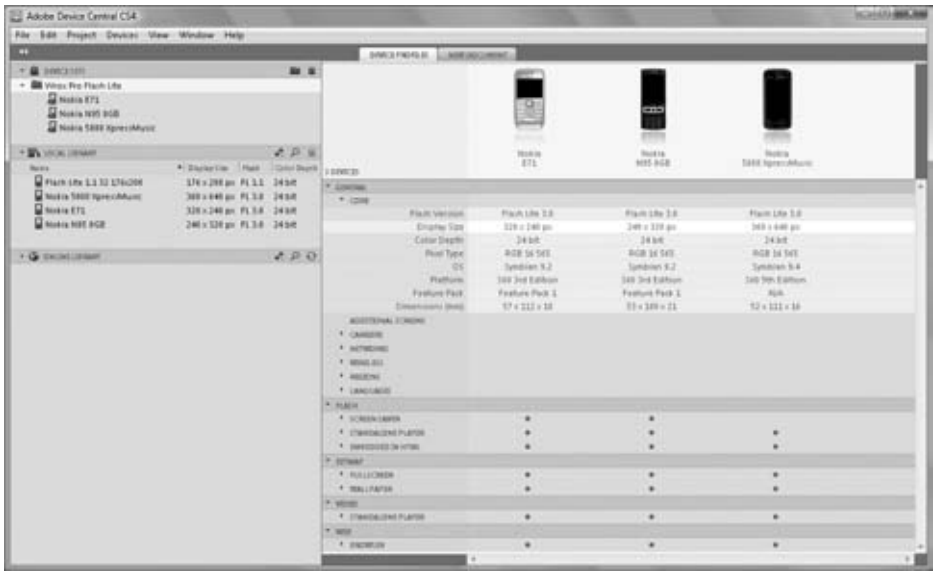


FIGURE 2-4

Note that each of the three devices has a different display size:

- Nokia E71 is 320 × 240 px
- Nokia N95 8GB is 240 is 240 × 320 px
- Nokia 5800 XpressMusic is 360 × 640 px

www.it-ebooks.info

You may think that the display size actually represents the viewable area of your Flash content, but this is actually not 100 percent the case. A few interrelated factors can determine the viewable area of your Flash content:

- Type of Flash content
- Addressable size
- Display mode

At this point you should note that creating content for one particular display size will not necessarily mean that content will display correctly on another device with a different display size. You must ensure that in developing content, whether it is a screensaver, wallpaper, browser, or standalone content, you are aware of the display sizes for targeted handsets. Throughout the chapters of the book you'll be developing applications under the Standalone Player content type.

The addressable size is actually the default area that the player occupies on the device when the display mode of the Flash Lite Player is in its normal state. This information can be located in each device's Device Profiles panel. Select a device, then Flash ➪ Standalone Player, to view this.

Table 2-1 displays the values for Flash Lite standalone player's addressable size alongside the display size for the three Nokia devices we've been working with.

TABLE 2-1

DEVICE	DISPLAY SIZE	ADDRESSABLE SIZE
Nokia E71	320 × 240 px	320 × 200 px
Nokia N95 8GB	240 × 320 px	240 × 238 px
Nokia 5800 XpressMusic	360 × 640 px	360 × 490 px

Notice that there is a marked difference between the display size and the addressable size, and that there is less space for viewable content in the latter. In these examples the height of the content is the differing property of the Flash movie. This would have the effect of distorting ("squashing") content developed for its related display size dimensions. For instance, the Nokia 5800 XpressMusic has a 150 px difference in height.

Modifying the Display Mode

For standalone content you can get the maximum out of a device's screen size by setting the display mode to full screen, and in Device Central you can set the display mode when you create a new Flash mobile document:

1. In Device Central, select File ➪ New Document In ➪ Flash.
2. In the New Document Panel that opens, ensure that the Set to Fullscreen checkbox is selected.

When you create your application, the application will automatically change the display mode to full screen. What this actually does is add a method called `fscommand2()` to the application when it is created.

www.it-ebooks.info



We'll cover the `fscommand2()` *method in Chapter 3. Here you just need to be aware that the viewable area for the application can be maximized on the device by setting the display mode to Fullscreen, and the ActionScript line* `fscommand2("FullScreen", "true");` *is added to the first frame on the timeline in the* `.fla` *file for your project.*

Creating New Flash Mobile Documents

There are actually two ways to create new Flash mobile documents. The first we've already mentioned: through the File ➪ New Document In ➪ Flash path. The second is through Adobe Flash CS4 Professional.

1. Open Flash CS4, then select File ➪ New and in the panel that opens select Flash File (Mobile). This will automatically launch Device Central if it isn't already open.

   Figure 2-5 shows the New Document panel in Flash CS4.



FIGURE 2-5

In Device Central you'll now see the New Document panel in focus. The panel allows you to specify the details of your new Flash mobile project. Here you can define the following:

- Player version
- ActionScript version
- Content type
- Fullscreen
- Custom size (width and height)

www.it-ebooks.info

The majority of the examples in this book have the player version set to Flash Lite 3.0. This is the version of the runtime commonly found as the pre-installed version on many Nokia devices found in Device Central. Each `.fla` file has already been created for you, so there's not really a need for you to go through the steps of creating each Flash mobile project from scratch. However, if you are creating the examples from scratch, the player version should be set to Flash Lite 3.0 in the drop-down list. You'll also need to ensure that the ActionScript version is ActionScript 2.0, content type is Standalone, Fullscreen is selected, and Custom Size is deselected.

While you can specify the properties for the document as described in the preceding paragraph, you can also choose to create a mobile project by selecting a device that you have in your device sets folder or local library to use as a starting point. Selecting multiple devices will group devices by size in the New Document panel. If you select multiple devices with different display sizes, you must decide which size to use for your `.fla` file (see Figure 2.6).

Figure 2-6 shows the New Document panel.



FIGURE 2-6

2. Click the Create button to create the `.fla` file with the document settings you have specified.

Device Profiles Used in this Book

Here's a list of the device profiles used in the book:

- **Nokia N95 8GB:** Chapters 3, 5, 7, 9, and 13
- **Nokia E71:** Chapters 2, 6, 8, and 13

www.it-ebooks.info

- **Nokia 5800 XpressMusic:** Chapters 10 and 11
- **Sony Ericsson C510:** Chapters 4 and 12

Before proceeding to the next section, add the Sony Ericsson C510 device profile to your Wrox device set. This device actually supports content targeted at player version Flash Lite 2.1 and is used for demonstrating examples created in Chapters 4 and 12.

CREATING A "HELLO WORLD" EXAMPLE

In this section you will create your first Flash mobile project from scratch using Flash CS4 and Device Central.

The task will be simply to create a "Hello World" example. Here you will display the "Hello World!" message in the application while you go through the steps of using the tools.

1. Open Flash CS4.
2. Create a new mobile project by selecting File ➪ New ➪ Flash File (Mobile). The focus will now move away from Flash CS4 to Device Central, which should start up in a new window. You'll return to Flash CS4 shortly.
3. In your Device Sets panel, select the Nokia E71. At this stage you should have already created a device set containing all the device profiles you'll need for the examples in this book; if you haven't, read the section "Creating Device Sets."
4. In the New Document panel that opens, ensure that Flash Lite 3.0 is selected as the player version, then select ActionScript 2.0 for the ActionScript version and ensure that the content type selected is standalone player.

   Figure 2.7 shows the Nokia E71 selected in the New Document view.

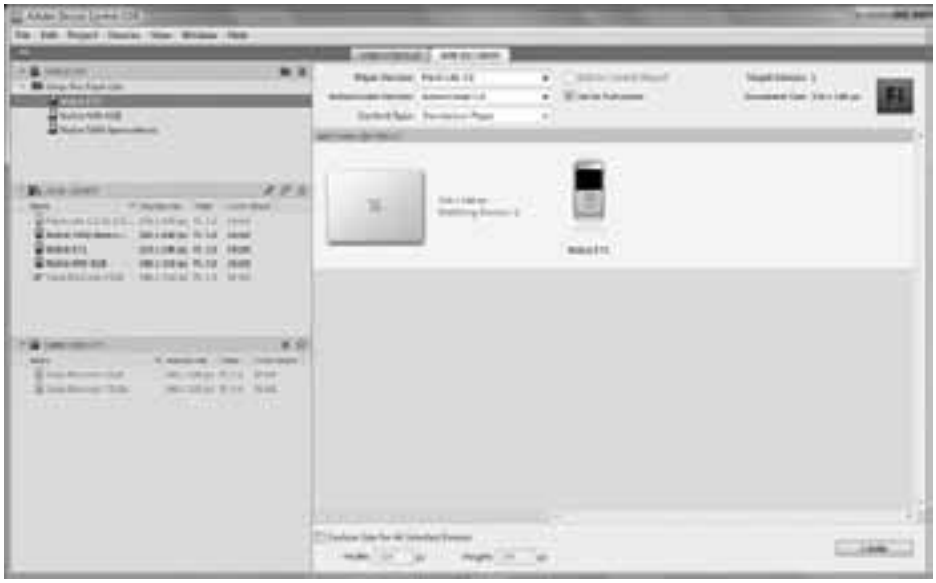


FIGURE 2-7

www.it-ebooks.info

5. Select the Set to Fullscreen checkbox to ensure that the maximum display area of the device will be used. Note that the Document Size will change from the addressable size, 320 × 200 px, to the display size of the device, 320 × 240 px. The Document Size should be 320 × 240 px.
6. Click the Create button at the bottom right of Device Central to create the new Flash mobile document. You now return to Flash CS4, where a new document has been created matching the settings used in Device Central.
7. In Flash CS4 you should notice that there is a layer in the timeline that has ActionScript code on the first frame. Select this layer and rename it Actions.
8. Select the first frame and press F9 to bring up the Actions panel, or select Window ➪ Actions. You should see the `fscommand2()` method to setting the Fullscreen display mode.
9. Next save the document. Select File ➪ Save As, and save the `.fla` file as `helloWorld.fla`.
10. Create a new layer for the TextField by selecting Insert ➪ Timeline ➪ Layer. Highlight the layer in the timeline and rename it Text.
11. Open the Tools panel. Press Ctrl+F2 or select Window ➪ Tools. Then select the Text tool to create a TextField at the center of the stage. Add some text to the TextField. "Hello World!" should be sufficient.
12. While the TextField is still selected, press Ctrl+F3 or select Window ➪ Properties to open the Properties panel.
13. Change the text type from Static to Dynamic. Then type **`helloWorld_txt`** in the Textfield for the instance name replacing <Instance Name>.
14. Still in the Properties panel for our text, select the Character submenu. In the Size field, enter **36**. Then in the Color field enter a value for the text color #009966.
15. With the Properties panel still open, note that underneath the Anti-alias drop-down are three buttons. The third sets the border around the TextField. Ensure that Show border around Text is disabled.

    Figure 2-8 shows the Properties panel for the TextField in `helloWorld.fla`.



FIGURE 2-8

16. Select Control ➪ Test Movie or press Ctrl+Enter to run the example. Focus now returns to Device Central.

In Device Central you will see that the emulator has started with the application running. Our "Hello World" message is displayed (see Figure 2-9).

www.it-ebooks.info



FIGURE 2-9

17. Return to Flash CS4, then select File ➪ Save.

Your first piece of Flash content, albeit a very basic piece, has now been created and can be found in the same directory as the `.fla` file you saved. The filename should be `helloWorld.swf`.

In Flash CS4 with your first project still open, let's take a look at Publish settings. Select File ➪ Publish Settings. In the window that opens you'll see a number of additional settings for how your content will be published.

Figure 2-10 shows the Publish settings panel for `helloWorld.fla`.

Here you'll find settings for images and sound, including JPEG quality, audio streaming, and local playback security, which ultimately can have an effect on how your content performs and previews. We'll cover each one of these throughout the course of the book.

You'll also notice in the Publish settings that Player and Script in the Publish settings are the same values you defined in Device Central. Next to the Script selection is a Settings button that allows you to



FIGURE 2-10

www.it-ebooks.info

define paths for class files. We'll cover this in the section "ActionScript 2.0 and Object-Oriented Development" in Chapter 3.

In the next section we take a closer look at the emulator, before looking at some of the features of Device Central.

USING THE EMULATOR IN DEVICE CENTRAL

Although we're not testing the limits of the Flash Lite Player with "Hello World," if you run the example again you should notice that a range of features can be found on the right side of the Device Central CS4 Emulator. These features allow you to alter the emulator's settings and manipulate variables that if they were changed on a real device would have an effect on the content you're developing. These features will help you in testing your Flash Lite content under the various settings. There is also other valuable information detailed in this section. Here's a list of all those sections with brief usages:

- **Content Type:** This section contains a drop-down list that allows you to change the content type for your Flash content. To see how your content would appear on devices supporting that type, simply select an option from the list.
- **File Info:** This section displays file information about your Flash content, including name, file size, Flash version, and dimensions.
- **Display:** This section allows you to modify settings that affect the device's display, including the backlight, timeout, reflections, gamma, contrast, screen mode, and fullscreen.

  The backlight is a feature on handsets that illuminates the display screen. In Device Central you can change the illumination and see how your content looks at various settings. The timeout setting works in conjunction with the backlight. When you have this checkbox selected, you can simulate the length of time before the backlight automatically turns off, just like on a real device. Figure 2-11 shows the Display panel in Device Central with the timeout value set to 4 seconds.



FIGURE 2-11

- **Automated Testing:** This section allows you to create automated tests for your Flash content. Here you record a run-through of your application to capture your button presses, key input, network requests and responses, and so on while outputting debug information. You can then re-run the procedures across multiple devices in your device sets sequentially, without having to key in the same button presses.
- **Key Pad:** This section provides you with an additional way to control your Flash content. It's actually a replication of the keypad used for the Adobe Generic Phone shown earlier.
- **Memory:** This section displays information on memory consumption during the life of your Flash content.

www.it-ebooks.info

- **Device Status:** This section allows you to modify the device emulator settings for language, time zone, date, time, volume, battery, and charger. Each of these properties can be retrieved using ActionScript, as you learn in Chapter 3.

  Figure 2-12 shows the Device Status panel in Device Central.



FIGURE 2-12

- **Device Performance:** This section allows you to alter the device's performance. Here for instance, you can modify settings to simulate how your Flash content runs.
- **Network Status:** This section allows you to change settings related to the network, including name, generation, status, connection, and signal.

  The signal strength displayed on devices indicates the network connectivity to enable the user to determine the call quality and the ability to receive phone calls or data. Devices don't always have maximum signal strength, so in this panel you can alter the signal strength to simulate the effect of running your application on a device when there is a poor or favorable signal. Figure 2-13 shows an example of the Network Status panel in Device Central with the signal strength set to 27 percent.



FIGURE 2-13

- **Network Performance:** This section allows you to modify variables such as network availability and download and upload speeds. The latency of a network connection can also be altered here.
- **Persistent Storage:** This section displays information on persistent storage, including total available bytes and the used or free bytes allocated on the device for the Flash Lite Player.
- **Security:** This section allows you to select between the default sandbox or local trusted sandbox Flash content.
- **Message:** This section displays error messages (shown only when there is an error) related to unsupported content.

Each of the categories listed has been specifically added to address common problems or situations that the mobile developer may encounter during development, and they are tools that aid in building applications by allowing you to simulate the environment of a real device, such as setting the battery level to low, changing the network status to one that is intermittent, monitoring the memory, and viewing the amount of persistent storage available to the Flash Lite Player.

Later in the book we'll take a look at additional aspects of the emulator that are helpful in Flash Lite mobile development. Here we consider memory and device CPU and take a look at the Memory and Device Performance panels in more detail.

Memory and Device CPU

As highlighted in Chapter 1, the Flash Lite Player is a version of the Flash Player used for desktops and browsers that is optimized for use on mobile devices. Such a dedicated version is needed because the majority of mobile devices are limited by the memory available and speed of their CPU,

www.it-ebooks.info

and these two factors largely have an impact on an application's performance on the device. It is vitally important that when developing applications for the Flash Lite Player you take this into consideration. Memory is a device constraint. Imagine a scenario in which you create an application that does all the magical things you require, but it leaves the device unable to handle incoming calls or text messages. Or, worse, your application crashes the device after only a few transitions of data consumption.

The performance of your application is affected by the content and the objects within it, hence applications must be written so that they consume as little memory and CPU as possible. For example, the minimum RAM requirement for the Flash Lite 3.1 Player is 128K, and the recommended requirement for standalone content is 2MB (which can also vary depending on whether the application has video included).

As a rule of thumb, Adobe has provided an estimated "worst-case memory consumption" for runtime playback of the Flash Lite Player. For version 3.1 it stipulates a content–heap size ratio of 1:15, hence, if the size of your final application file is 100KB, Adobe's recommendation is that the memory configuration for the device be set to at least 1.5MB.

It's also possible that some devices which have the Flash Lite Player installed won't provide you with the best environment to run your application. The device OEMs work with Adobe to accommodate the Flash Lite Player, allocating an upper limit of memory that can vary from device to device.

Within Device Central, the two features that can aid in monitoring and testing your application's performance and memory consumption are the following, as mentioned earlier:

- Memory panel
- Device Performance panel

Using the Memory Panel

Open an application in Device Central and then navigate to the Memory panel via the Emulator tab. The Memory panel indicates how much memory your Flash Lite application is consuming during playback by displaying the device memory information in graphical form.

This is a powerful tool for monitoring what happens during the life of your application, and during development it provides you with a very important point of reference for feedback that otherwise would be available only by testing your application on the actual device. If you are targeting more than one device, check the memory consumption graph while running your application on each device profile through the emulator. This will help you determine the importance of a transition, or animation; see what happens when assets are attached from the library; and help you realize how much or little content loaded from external sources consumes memory.

Figure 2-14 shows the Memory panel running in Device Central.

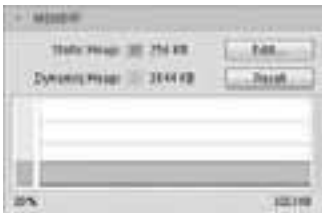


FIGURE 2-14

The key on the graph shown in Figure 2-14 indicates two types of memory allocation for the device:

- **Static Heap:** This is a fixed-size memory allocator reserved by the Flash Lite Player when the player and your application

www.it-ebooks.info

initialize. All devices that have the Flash Lite Player installed will have a static heap. Static heap memory is not released to the operating system of the device once it has been used.

- **Dynamic Heap:** This is a variable-sized memory allocator not reserved by the Flash Lite Player, initialized when a request for memory is sent to the player. Not all devices that have the Flash Lite Player installed will have a dynamic heap. The dynamic heap has a limited allocation based on the device's operating system and defined by the OEM. Dynamic heap can be released to the operating system of the device and can increase or decrease at runtime.

Heap is the size of the allocated memory determined at runtime, used when the amount of memory needed is considered to be unknown and variable as it may increase depending on the needs of the application. Memory for the Flash Lite Player is determined by the operating system for the device. In another form of memory allocation the stack is determined at compile time.



One significant value to remember is 32KB. This value represents the size of the data allocated for every object that requests memory via the Flash Lite Player to the operating system.

It is important to remember that memory consumption can grow unexpectedly during the life of a Flash Lite application. When the application exceeds the combined static heap and dynamic heap values, an error is thrown and your application effectively crashes.

Figure 2-15 shows how the dynamic heap grows during the life of an application.

Each of the following areas affects how the Flash Lite Player manages certain aspects of memory as shown in Figure 2-15:

- Rendering graphics
- Embedding fonts
- Loading files, images, SWF, text, and XML
- Utilizing audio and video



FIGURE 2-15

Using the Device Performance Panel

You can effectively replicate the performance of your application on the device by using the Device Central emulator and the Device Performance panel. Figure 2-16 shows the Device Performance panel in Device Central.

As you can see from the figure, the panel has several field settings each with their own point of usage:

- **Calibrate:** Click this button to effectively activate simulations for the target emulated device profile. Device Central has a standard performance calibration `.swf` file



FIGURE 2-16

www.it-ebooks.info

that it runs across all devices in its database as a benchmark. You must run a calibration before running a simulation.

- **Index value:** This value indicates the performance of the device when compared with the benchmark performance calibration. The higher this value, the higher performance is expected by the device.
- **Speed:** The slider and input box allow you to set the percentage execution speed of the device, which is effectively the processing performance. The lower the value, the slower the execution speed of the application.
- **Rendering quality:** This drop-down allows you to see how changing the quality setting affects the performance of the application during a simulation. Flash Lite has three quality settings: low, medium, and high. The higher the quality setting, the more accurately and smoothly the Flash Lite Player renders vector outlines; the lower-quality setting results in less smoothly drawn outlines. The rendering quality can affect animation performance and frame rate, and how text is rendered. The quality setting is another property that can be set using the `fscommand2()` method via ActionScript. This is covered in Chapter 3.
- **Simulate Performance:** When this checkbox is selected, the application will run a simulation in the emulator with the pre-mentioned settings as described.

The aim of the Device Performance panel is to provide you with the ability to test your application under variable device conditions to simulate a real device. The performance of a device can easily be affected by other processes running, so this panel allows you to see how your content will perform by changing some of the settings listed.

The most important thing to remember about emulation and using the Device Performance panel in Device Central is that it helps you to effectively test how your content will perform under different conditions.

TESTING ON MOBILE DEVICES

Using the Device Central emulator while creating applications is great for previewing content on devices that you don't have, but before you deliver your applications to consumers it is highly recommended that you test your content on actual mobile devices.

You have a few other options available for testing on devices including:

- Devices with Flash Lite pre-installed
- Remote devices

One prerequisite to the testing of Flash Lite content on devices is that they must have a version of the Flash Lite Player installed. You will need to ensure that the player version you have targeted during development is equal to or lower than the version of the player on the device. You can use Device Central as an aid to determining what version of the Flash Lite runtime runs on a device.

www.it-ebooks.info

Devices with Flash Lite Pre-Installed

The easiest approach by far for testing content is on a device with Flash Lite pre-installed. It's highly recommended that you obtain a device with the latest version of the Flash Lite runtime pre-installed as doing so will make testing much easier. For testing content in development you can simply transfer your `.swf` files to a device via Bluetooth or USB straight from your working folder.

Remote Devices

An alternative solution to testing your Flash Lite content on a physical device is to use a remote device online.

Remote Device Access (RDA)

The Remote Device Access (RDA) service at `http://apu.ndhub.net/`, supported by Forum Nokia, is one option that allows developers to gain access to a range of Nokia devices over the Internet. This is a live online service where you reserve handsets for a period of time allowing you to remotely test packaged Flash Lite applications on devices that you would have to otherwise pay for or borrow.

DeviceAnywhere

With DeviceAnywhere (`http://www.deviceanywhere.com`) you can also test and run your Flash Lite content on remote devices online. The service currently provides a collection of handsets from numerous manufacturers. You can also run content on devices with different operators.

SUMMARY

In this chapter we looked at the tools needed for creating Flash Lite mobile applications using Flash CS4 and Device Central CS4.

You learned how to import device sets into Device Central CS4 and how to compare multiple device profiles and features.

With the "Hello World" example you created, you also learned how to create new Flash mobile projects.

Following on from your first project, you examined the core features of the Device Central CS4 emulator and gained an understanding of the memory and CPU considerations for developing Flash Lite content.

Finally, you were given tips on testing content on physical and remote devices.

In Chapter 3 we'll take a look at ActionScript and object-oriented programming development used for Flash Lite development, drawing on some of the topics touched on in this chapter, such as creating classes, `fscommand2()` method calls, and device status properties.

www.it-ebooks.info

3

Object-Oriented Programming for Flash Lite Development

WHAT'S IN THIS CHAPTER?

- Object-oriented programming concepts
- Creating extensible and reusable classes
- Using fscommand2()
- The System.capabilities object
- Creating events and event listeners
- Handling key, button, and touch input

This chapter is about preparing you for programming your Flash Lite mobile applications. Here, you'll get an overview and refresher of the ActionScript 2.0 (AS2) language, along with code snippets and more detail on the optimized version of the Flash Player for mobile devices Flash Lite 3.*x*.

You'll cover the fundamental aspects of AS2 required for developing Flash Lite mobile device applications. You will go through the key considerations and coding aspects that you need to bear in mind along the way.

AS2 REVISITED

The current version of the ActionScript programming language is AS3. With the majority of *Flash-enabled* devices utilizing Flash Lite, the focus on creating incredible mobile applications lies with the older brother AS2 to develop for devices, hence the need to visit key aspects of the language.

www.it-ebooks.info

The AS2 programming language has been supported by the Adobe Flash Platform since 2003. It was compiled for byte code use in Flash Player 7, a significant feature in the release of Macromedia's Flash MX 2004 IDE. AS2 provided new ways for Flash developers to utilize complex code and create compelling web-based applications that the developers in the community had longed for. The evolution of the language from AS1 was considered to be a huge step in the right direction for the developer community. Among its features provided the long-awaited formal and structured approach to development in the form of new syntax and object-oriented programming (OOP), which included full support for classes, inheritance, interfaces, and other OOP concepts.

AS2

AS2 is based on the international standard ECMA-262, the ECMAScript Language Specification (ECMA, `http://www.ecma-international.org/publications/standards/Ecma-262.htm`). ECMA stands for European Computer Manufacturers Association).

The following elements and keywords, which you should already be familiar with, were introduced to Flash Player 7 for AS2:

- `class` — Keyword to declare a new data type and object *blueprint*, not predefined in the AS2 language.
- `extends` — Keyword to denote the class is inheriting properties and methods from another class.
- `interface` — Keyword to declare a list of properties and methods which a class should implement.
- `implements` — Keyword to denote that a class will need to implement all the declarations found in an interface.
- `dynamic` — Keyword to define a class that can have properties and methods added to a class at runtime, that aren't defined in the original class definition.
- `static` — Keyword to declare a property or method of a class that can be referenced or run without needing to create an instance of the class.
- `public` — Keyword used to define an open scope access declaration for variables and methods.
- `private` — Keyword used to define a restricted scope declaration, for variables and methods.
- `get` — Keyword used to declare methods that must return a value.
- `set` — Keyword used to declare that a function must set a value.
- `import` — Keyword used to define the name of a class you have defined in an external class file or the directory in which you have stored related class files.

www.it-ebooks.info

Throughout this chapter, and the remainder of the book, you'll cover many examples of these keywords.

OBJECT-ORIENTED PROGRAMMING FOR FLASH LITE

In this section you will get an overview of object-oriented programming (OOP) in the context of developing for Flash Lite.

Why OOP?

The importance of using custom classes for development is one that should not be overlooked. In fact, class-based development is imperative if you are to stand any chance of creating and maintaining complex mobile device applications for Flash Lite.

Not only does class-based development allow you to create new data types that are not predefined in the ActionScript language, you can also leverage existing core classes by extending them. Used correctly there are also the fundamentals of OOP, which as you learn, also add value to your development time.

There are three universally adopted criteria which are important concepts of OOP in software development:

- `Polymorphism` — The ability to process objects in different ways, depending on their data type or class and redefine methods for derived classes.
- `Inheritance` — The ability to share some or all of another class's main characteristics, properties, and methods.
- `Encapsulation` — The ability for an instance of a class to be used without the need to know about its properties and inner workings.

These criteria are what make OOP incredibly productive and reusable and are crucial principals to be aware of in any software development.

In the next section you will go through an example of applying OOP to a real-world example covering each of the concepts just mentioned.

An OOP Example for Mobile

Take your mobile device as an example of a real-world object. It is a universal communications tool, and an object that can exist in many forms, but generally has similar physical features across the world, which when analyzed and compared with other mobile devices have properties, features, and functions that are *generic*.

In Table 3-1 you'll see a comparison of some of the properties and features of two mobile devices, the Sony Ericsson Aino and the Nokia E71.

www.it-ebooks.info

TABLE 3-1: A Comparison of some of the properties and features of the Sony Ericsson Aino and Nokia E71

DEVICE PROPERTIES AND FEATURES	SONY ERICSSON AINO	NOKIA E71
Manufacturer	Sony Ericsson	Nokia
Model	Aino	E71
Size	4.1 x 2.0 x 0.6 inches	4.49 x 2.24 x 0.39 inches
Weight	134.0 g	127.0 g
Screen	432 x 240 px	320 x 240 px
Platform	Sony Ericsson Java Platform 8.5	S60 3rd Edition, Feature Pack 1 based on Symbian OS v9.2
Available colors	Luminous white, obsidian black	Gray steel, white steel
Input	Touch UI, keypad	QWERTY keypad
Communication	Voice, Internet, email, video conferencing	Voice, Internet, email, video conferencing
Special features	Accelerometer, GPS	GPS

Creating a Class

Using some of the properties from Table 3-1, you could create a class for the Sony Ericsson Aino mobile device; let's call it `Aino`.

The class is defined with class name, package name, and class constructor (see Listing 3-1). The constructor is `public`, which will allow you to create an instance of the class. The package name for `Aino` is defined as `com.wrox.chapter3`, which ultimately translates to a file path "com/wrox/chapter3," where you'll find the `Aino` class.



Where classes have been specially created for the book, I have used the same reverse domain convention "com/wrox/chapter number." This should make it easier for you to find the code that relates to each chapter.



LISTING 3-1: The class name, package name and constructor for Aino

```
class com.wrox.chapter3.Aino {

      public function Aino(){

      }
}
```

www.it-ebooks.info

In Listing 3-2 you'll see two private properties that have been declared for the `Aino`, called `manufacturer` and `model`; their values are also defined, having been taken from Table 3-1. Because we know the Aino device has these properties, we can set the values for them in the constructor, as they won't necessarily need to change when an object instance is created.



LISTING 3-2: The manufacturer and model properties for Aino

```
class com.wrox.chapter3.Aino {

      private var manufacturer:String;
      private var model:String;

      public function Aino(){

            manufacturer = "Sony Ericsson";
            model = "Aino";

            trace("Aino created!);

      }

}
```

To create an instance of the `Aino` class, you can simply call the following AS in the Timeline (see Listing 3-3).



LISTING 3-3: Creating an instance of Aino

```
var myAino:Aino = new Aino();
```

At the moment, there is no way to retrieve or set any of the values on an instance of `Aino`. To do this, you can add getters and setters for each property. As shown in Listing 3-4, the `Aino` class has four public methods to set and retrieve the values for these properties. The `get`ter methods for manufacturer and model return their respective property values, which are typed as `String` variables. The `set`ter methods for manufacturer and model alter properties and don't need to return a value, so these method have return types defined as `Void`.



LISTING 3-4: The getter and setter methods for Aino

```
class com.wrox.chapter3.Aino {

      private var manufacturer:String;
      private var model:String;

      public function Aino(){

            manufacturer = "Sony Ericsson";
```

continues

www.it-ebooks.info

LISTING 3-4 *(continued)*

```
            model = "Aino";
        }

        public function get manufacturer():String {
            return manufacturer;
        }

        public function set manufacturer(value:String):Void {
            manufacturer = value;
        }

        public function get model():String {
            return model;
        }

        public function set model(value:String):Void {
            model = value;
        }
}
```

You can now set the values for each property after you've created an instance `Aino` (see Listing 3-5).



LISTING 3-5: Setting the properties for the Aino instance

```
var myAino:Aino = new Aino();
myAino.manufacturer = "Sony Ericsson";
myAino.model = "Aino";
```

While the code seems perfectly reasonable so far, there are two minor problems with the approach in the `Aino` class. The first is that both manufacturer and model properties are defined in the constructor specifically for the creation of `Aino` objects, which is fine for creating `Aino` instances, but what about the E71 *and other mobile devices?* Of course, you can now set the properties for the instances to account for these, but what about other features that may be described in the class that we haven't discussed, such as the presence of Touch UI and QWERTY keys? Also the manufacturer and model shouldn't really be set after creating the `Aino` class, because these properties are what actually define it; more on this later.

Ultimately, the properties and functions of the `Aino` class could form the base blueprint of every mobile device on the market. Having tabulated examples of what distinguishes one mobile device handset from another, with properties such as: manufacturer, name, model, size, and weight, the blueprint for the class should actually be part of something more *generic*.

To make life easier, you can use class diagrams to theorize code and map out how objects relate to each other. For this example, we want to create a class that is generic to *Mobile* object creation.

Figure 3-1 is a class diagram showing a relationship between a base class called `Mobile`; two classes, `Aino` and `E71`; and an interface called `IMobile`. Class diagrams should be based on real-world

www.it-ebooks.info

scenarios; for the purpose of modeling these two mobile devices, not all features have been included for these two mobiles in particular.

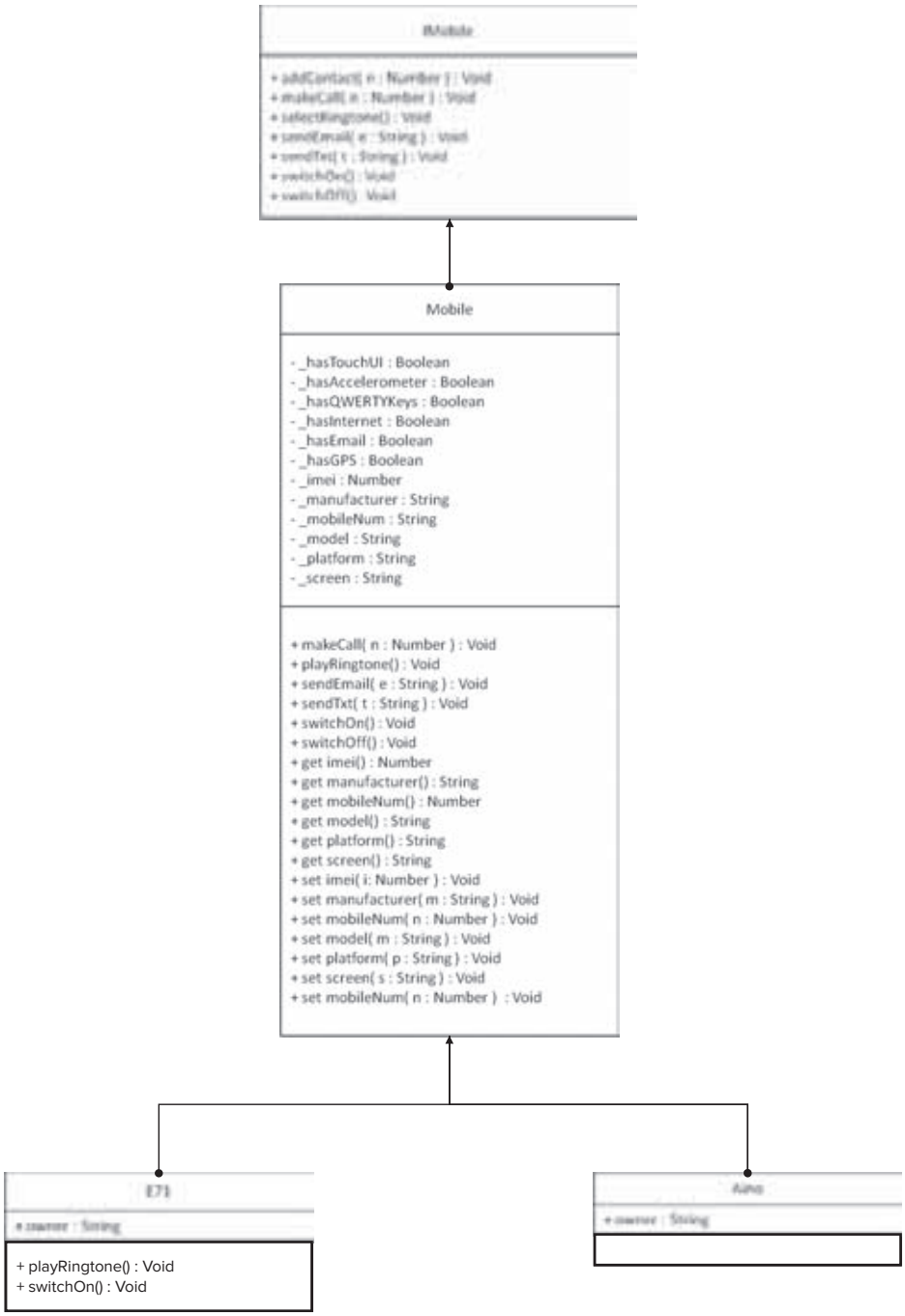


FIGURE 3-1

www.it-ebooks.info

The next few sections will cover each of the aspects in Figure 3-1.

Creating an Interface

Let's start with `IMobile`. This is an `interface`, a file that will define all the methods that the base class for mobile devices should implement. In `IMobile` there are six public method declarations:

- `makeCall()` — A public method to allow a mobile device to call another device. This is a feature undoubtedly common in all mobile devices.
- `playRingtone()` — A public method to allow a mobile device to play a ringtone. Again common to all mobile devices, this is something that we'll be able to distinguish across some devices.
- `sendEmail()` — A public method to allow a mobile device to send email.
- `sendTxt()` — A public method to allow a mobile device to send a text message.
- `switchOn()` — A public method to allow a mobile device to be switched on. Most devices have manufacturer's logos that appear when a device turns on, before someone actually gets to use it, so we can also use this as another way of distinguishing between devices.
- `switchOff()` — A public method to allow a mobile device to be switched off. Logos can sometimes appear before device shutdown also.

There are obviously more mobile-specific functions we could add to the method declarations, but for the `IMobile` example, we'll simply use those listed (see Listing 3-6). Variable declarations are not permitted in interfaces, and neither are getter and setter methods; private methods also cannot be specified.



LISTING 3-6: The IMobile interface

```
interface com.wrox.chapter3.IMobile {
      public function makeCall(n:Number):Void;
      public function playRingtone():Void
      public function sendEmail(e:String):Void
      public function sendTxt(t:String):Void
      public function switchOn():Void
      public function switchOff():Void
}
```

Listing 3-7 shows the example base class `Mobile`, which implements all the methods of the `IMobile` interface. In `Mobile` you'll find the following 12 private properties declared, as shown in Figure 3-1:

- `_hasTouchUI` — For indicating whether or not a mobile device has a touch screen interface for user input.
- `_hasAccelerometer` — For indicating whether or not a mobile device has an accelerometer.
- `_hasQWERTYKeys` — For indicating whether or not a mobile device has a QWERTY keyboard.

www.it-ebooks.info

- `_hasInternet` — For indicating whether or not a mobile device is capable of accessing the Internet.
- `_hasEmail` — For indicating whether or not a mobile device is capable of sending email.
- `_hasGPS` — For indicating whether or not a mobile device has GPS capabilities.
- `_imei` — For storing a value for the IMEI number.
- `_manufacturer` — For storing a value for the mobile device manufacturer.
- `_mobileNum` — For storing a value for the mobile device's mobile number.
- `_model` — For storing a value for the mobile device model number.
- `_platform` — For storing a value for the mobile device platform.
- `_screen` — For storing a value for the mobile device screen property.

In addition to these private variables, you'll notice that there are numerous getter and setter methods for each of these properties, and to instantiate the class, the manufacturer `m1`, model `m2`, and platform `p` properties need to be specified; this is a significant point to note (see Listing 3-7). Again, there are obviously more mobile-specific properties we could use in the example. However, there's enough here to continue with explaining the concept. The `imei` property specified represents a unique identifier for mobile devices.



LISTING 3-7: The Mobile class with all properties and methods defined

```
import com.wrox.chapter3.IMobile;

class com.wrox.chapter3.Mobile implements IMobile {

      private var _hasTouchUI:Boolean;
      private var _hasAccelerometer:Boolean;
      private var _hasQWERTYKeys:Boolean;
      private var _hasInternet:Boolean;
      private var _hasEmail:Boolean;
      private var _hasGPS:Boolean;
      private var _imei:Number;
      private var _manufacturer:String;
      private var _mobileNum:Number;
      private var _model:String;
      private var _platform:String;
      private var _screen:String;

      public function Mobile(m1:String, m2:String, p:String) {
            _imei = random(16); /* Press "*06*#" on the device */
            _manufacturer = m1;
            _model = m2;
            _platform = p;
      }

      public function makeCall(n:Number):Void {
            trace(" .dialing " + n.toString() + " FROM " + mobileNum.toString() );
```

continues

www.it-ebooks.info

LISTING 3-7 *(continued)*

```
}

public function playRingtone():Void {
      trace(".playing the default ringtone.");
}

public function sendEmail(e:String):Void {
      trace(" .sending an EMAIL: " + e);
}

public function sendTxt(t:String):Void {
      trace(" .sending a TEXT: " + t);
}

public function switchOn():Void {
      trace(" .mobile is ON! ");
}

public function switchOff():Void {
      trace(" .mobile is OFF! ");
}

public function get hasTouchUI():Boolean {
      return _hasTouchUI;
}

public function set hasTouchUI(b:Boolean):Void {
      _hasTouchUI = b;
}

public function get hasAccelerometer():Boolean {
      return _hasAccelerometer;
}

public function set hasAccelerometer(b:Boolean):Void {
      _hasAccelerometer = b;
}

public function get hasQWERTYKeys():Boolean {
      return _hasQWERTYKeys;
}

public function set hasQWERTYKeys(b:Boolean):Void {
      _hasQWERTYKeys = b;
}

public function get hasInternet():Boolean {
      return _hasInternet;
}

public function set hasInternet(b:Boolean):Void {
      _hasInternet = b;
```

www.it-ebooks.info

```
}

public function get hasEmail():Boolean {
      return _hasEmail;
}

public function set hasEmail(b:Boolean):Void {
      _hasEmail = b;
}

public function get hasGPS():Boolean {
      return _hasGPS;
}

public function set hasGPS(b:Boolean):Void {
      _hasGPS = b;
}


public function get imei():Number {
      return _imei;
}

public function set imei(i:Number):Void {
      _imei = i;
}

public function get manufacturer():String {
      return _manufacturer;
}

public function set manufacturer(m:String):Void {
      _ manufacturer = m;
}

public function get mobileNum():Number {
      return _mobileNum;
}

public function set mobileNum(n:Number):Void {
      _mobileNum = n;
}

public function get model():String {
      return _model;
}

public function set model(m:String):Void {
      _model = m;
}

public function get platform():String {
      return _platform;
```

continues

www.it-ebooks.info

LISTING 3-7 *(continued)*

```
        }

        public function set platform(p:String):Void {
              _platform = p;
        }

        public function get screen():String {
              return _screen;
        }

        public function set screen(s:String):Void {
              _screen = s;
        }
}
```

You can now use the properties and methods of the `Mobile` class as a base, by extending it. Returning to the `Aino` class, you can inherit all the methods and properties of `Mobile`. In Listing 3-8 you'll see a few changes to the `Aino` class covered earlier. First, the class declaration now `extends` the `Mobile` class, and to do this the Mobile class is imported into `Aino`. Second, in the body of the constructor for `Aino`, you'll see it calls the `super()` function, passing it three values: the manufacturer, "Sony Ericsson"; the model, "Aino"; and the platform, "Sony Ericsson Java Platform 8.5". The `super()` function must be the first method called when you want to call the parent class method first. In this example, the constructor for `Aino` will call `Mobile()` constructor, which if you recall accepts three parameters. In doing this the properties for the Aino mobile device are set.



LISTING 3-8: The Aino class, extending Mobile and calling its constructor

```
import com.wrox.chapter3.Mobile;

class com.wrox.chapter3.Aino extends Mobile {

        public var owner:String;

        public function Aino(mobile:Number) {
              super("Sony Ericsson", "Aino", "Sony Ericsson Java Platform 8.5");
              mobileNum = mobile;
              hasAccelerometer = true;
              hasTouchUI = true;
              hasQWERTYKeys = false;
        }
}
```

The `Aino` class has a public variable called `owner`. As depicted in the class diagram, this allows you to assign an owner for the mobile device once it has been created. The `Aino` constructor itself takes a single parameter, called `mobile`, which is assigned to the `mobileNum` property inherited by the class. This is really here to illustrate that a mobile device should actually have a mobile number

www.it-ebooks.info

associated with it, when it is created in our scenario. Obviously, in the real world, mobile devices have SIM cards with mobile numbers associated with them, but for simplicity this is not modeled here. The `hasAccelerometer` and `hasTouchUI` properties in the constructor are set to `true`, as these are particular features of the Aino device, while `hasQWERTYKeys` is set to `false`. Other properties could be set here, based on the details listed in Table 3-1. The `mobileNum` property can be altered and retrieved through its getter and setter method, which is something that can actually happen in the real world. However, because the getter and setter functions have been declared for each property in the `Mobile` class, this poses a potential problem in that, once an `Aino` instance has been created, certain properties can be altered that in the real world wouldn't change, such as the model or manufacturer. In Listing 3-9 you'll see modifications to the setter methods for manufacturer and model properties in the `Mobile` class, which prevent the values from being changed; this, of course, could also be done for `hasAccelerometer`, `hasTouchUI`, and `hasQWERTYKeys`.



LISTING 3-9: Updating the manufacturer and model setter methods in Mobile

```
public function set manufacturer(m:String):Void {
      if (_manufacturer == null || _manufacturer == ""){
            _manufacturer = m;
      } else {
            trace("The manufacturer is already defined.");
      }
}

public function get mobileNum():Number {
      return _mobileNum;
}

public function set mobileNum(n:Number):Void {
      _mobileNum = n;
}

public function get model():String {
      return _model;
}

public function set model(m:String):Void {
      if (_model == null || _model == "") {
            _model = m;
      } else {
            trace("The model is already defined.");
      }
}
```

Having completed the `Mobile` class, you can use this to create other mobile devices, such as the Nokia E71. Listing 3-10 shows the `E71` class, which you can see also calls the `super()` function in the body of the constructor while setting different values each of the properties defined. Notice that the `playRingtone()` and `switchOn()` methods are overridden specifically for the E71 class; this was also specified in the class diagram earlier.

www.it-ebooks.info



LISTING 3-10: The E71 class, overriding the switchOn and playRingtone methods

```
import com.wrox.chapter3.Mobile;

class com.wrox.chapter3.E71 extends Mobile {

      public var owner:String;

      public function E71(mobile:Number) {
            super("Nokia", "E71", "S60 3rd Ed. FP1, Symbian OS v9.2");
            mobileNum = mobile;
            hasAccelerometer = false;
            hasTouchUI = false;
            hasQWERTYKeys = true;
      }

      public function playRingtone():Void {
            trace(".playing the Nokia theme.");
      }

      public function switchOn():Void {
            trace(".displaying Nokia welcome message.");
      }
}
```

Listing 3-11 shows how these classes can be used and interact with each other.



LISTING 3-11: The E71 class, overriding the switchOn and playRingtone methods

```
import com.wrox.chapter3.E71;
import com.wrox.chapter3.Aino;

var device1:Aino = new Aino(449876543210);
device1.owner = "Jermajesty";

trace("DEVICE 1:");
trace(device1.manufacturer);
trace(device1.model);
trace(device1.platform);

device1.switchOn();
device1.playRingtone();

var device2:E71 = new E71(440123456789);
device2.owner = "Jermaine";

trace( newline + "/————————————————/ " + newline );
trace("DEVICE 2:");
trace(device2.manufacturer);
trace(device2.model);
trace(device2.platform);

device2.switchOn();
```

www.it-ebooks.info

```
device2.playRingtone();

trace( newline + "/————————————————/" + newline );
trace("MAKE CALL FROM DEVICE 1 TO 2");

device1.makeCall(device2.mobileNum);
```

The code in Listing 3-11 is on the first frame of the Timeline in `chapter3_OOP.fla` file. When you run the example, open the Output panel in Device Central, select Window ➪ Flash Output. Figure 3-2 shows the trace output that is displayed.

This wraps up our example. Later you'll cover more mobile-specific features of the ActionScript language, which actually demonstrates retrieving features and properties of a device when content is actually running.



FIGURE 3-2

MOBILE DEVICE CONSIDERATIONS

Having covered OOP, the following sections will provide you with some aspects of your mobile application development that you need to consider when developing for Flash Lite using AS2.

Using fscommands to Communicate with the Device

Here, we discuss how you can utilize the native properties of a real mobile device through the Flash Lite player, using the global *fscommands* functions and combined leverage for your applications.

You will have several scenarios and subtopics to demonstrate why these functions are really important for you to learn, ultimately helping you to provide a more complete user experience for Flash mobile applications.



Native properties are those that are housed on and local to the device, such as network status and battery life.

There are two *fscommands*: `fscommand()` and `fscommand2()`. Each takes in a predefined "command," a string-based parameter that defines a message query to be sent to the Flash Lite player.

- `fscommand()` — Used largely for communication between the Flash Lite player and other applications on the device.
- `fscommand2()` — Used for interacting with device properties through the Flash Lite player and returns a value based on the query.

www.it-ebooks.info

fscommand()

For Flash Lite applications, the `fscommand()` call is limited in its applicability to launching external applications that exist on the mobile device, passing in parameters.

Here is an example of the `launch` command in action, opening a browser and setting the destination of the browser to the mobile version of the Twitter site:




```
fscommand("launch", "browser.exe, http://m.twitter.com");
```

code snippet chapter3_fscommand.as

It's important to note that `fscommand()` returns `Void` when executed, which makes it difficult to know whether it has run successfully. This is another one of the subtle differences between the two *fscommands*. This command always tends to be more effective when triggered by a button press.

fscommand2()

As previously mentioned, the `fscommand2()` call, in contrast to `fscommand()`, allows you to retrieve or set information, and also allows you to access features available on the device, which you can choose to include as part of your applications. In Chapter 2 you covered some of these aspects when you looked at Device Central. In this section you'll learn how you can utilize `fscommand2()` calls in the following scenarios:

- Indicating the device's battery status
- Displaying the network signal level
- Monitoring device memory allocation
- Handling network requests and connections

Indicating the Device's Battery Status

The `fscommand2()` method provides you with the facility to display to the user information about the battery level of the device during the life of the application. This is an important feature of AS2 and Flash mobile content, as it gives the user a good indication of battery life and can warn users that they may need to charge their device, preventing the device from completely dying unexpectedly, and hence demonstrating a little more thought in the user's experience.

Separate instances of the `fscommand2()` must be created to handle battery statuses, requiring one of the following *commands* to be supplied as an parameter:

- `GetMaxBatteryLevel` — To get the maximum battery level available on the device.
- `GetBatteryLevel` — To get the current battery level of the device.
- `GetPowerSource` — To get the power source. This indicates whether a device is attached to a power supply.

The following `Battery` class demonstrates how to determine the battery power remaining on the device and, if the remaining power is less than 10% and the power source is not external to the device, the user is prompted to connect a battery charger (see Listing 3-12).

www.it-ebooks.info



LISTING 3-12: The Battery class

```
class com.wrox.chapter3.Battery {

      public static var FSCOMMAND2_NOT_SUPPORTED:Number = -1;

      private var _currentLevel:Number;
      private var _maxLevel:Number;
      private var _warningLevel:Number;
      private var _isCharging:Boolean;

      public function Battery(){
            _currentLevel = fscommand2("GetBatteryLevel");
            _maxLevel = fscommand2("GetMaxBatteryLevel");
            _warningLevel = 10;
      }

      public function determinePowerRemaining():Void {

            switch(_currentLevel){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  default:
                        trace("_currentLevel = " + _currentLevel);
                  break;
            }

            switch(_maxLevel){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  default:
                        trace("_maxLevel = " + _maxLevel);
                  break;
            }

           isBelowWarningLevel();
      }

      private function fscommand2NotSupported():Void {
            trace("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
      }

      private function isBelowWarningLevel():Void {
            if(Math.round(_currentLevel/_maxLevel*100) < _warningLevel){
                  if(isCharging()){
                        trace("You are charging. It's OK!");
                  } else {
                        trace("Battery Level is LOW. Plug in a charger.");
                  }
            } else {
                  trace("Battery Level is FINE. ");
            }
```

continues

www.it-ebooks.info

LISTING 3-12 *(continued)*

```
        }

        private function isCharging():Boolean {
              isCharging = fscommand2("GetPowerSource");
              return _isCharging;
        }

        public function get warningLevel():Number {
              return _warningLevel;
        }

        public function set warningLevel(level:Number):Void {
              warningLevel = level;
        }

}
```

In the `chapter3_determiningBatteryLevel.fla` file, the `Battery` class is imported and the following code snippet is called on the first frame of the Timeline:




```
import com.wrox.chapter3.Battery;

var myBattery:Battery;
myBattery = new Battery();
myBattery.warningLevel = 10;
myBattery.determinePowerRemaining();
```

Code Snippet chapter3_determiningBatteryLevel.fla

You should see the trace statements that are written in the `Battery` class in the Output panel in Device Central. If you open the Device Status panel to the right of the emulator window, you can test and play around with the Battery level and then restart the example by pressing the "Go to beginning" button and then "Play." You should see in the output that, each time the example runs, the trace statements appear. This is a great way to test values while developing your applications.

Figure 3-3 shows the results from three consecutive runs of the example, when the battery is okay, when the battery is low, and when the battery is charging.



FIGURE 3-3

www.it-ebooks.info

Displaying the Network Signal Level

It's important to keep users in touch with a mobile device's main functions. One of the common visual representations across all mobile devices is the network operation signal level indicator and the strength of that signal. There are two calls via `fscommand2()` that allow you to measure the strength of the current signal, which will help you to determine the strength of a user's connection to a network without having to quit the app:

- `GetSignalLevel` — To get the current signal level.
- `GetMaxSignalLevel` — To get the maximum signal level value.

Listing 3-13 demonstrates an example of using `fscommand2()` to provide a universal mobile device signal.



LISTING 3-13: The Signal class

```
class com.wrox.chapter3.Signal {

      public static var FSCOMMAND2_NOT_SUPPORTED:Number = -1;

      private var _currentLevel:Number;
      private var _maxLevel:Number;
      private var _warningLevel:Number;

      public function Signal(){
            _currentLevel = fscommand2("GetSignalLevel");
            _maxLevel = fscommand2("GetMaxSignalLevel");
            _warningLevel = 10;
      }

      public function determineSignalStrength():Void {
            switch(_currentLevel){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  default:
                        trace("_currentLevel = " + _currentLevel);
                  break;
            }

            switch(_maxLevel){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  default:
                        trace("_maxLevel = " + _maxLevel);
                  break;
            }

            isBelowWarningLevel();
```

continues

www.it-ebooks.info

LISTING 3-13 *(continued)*

```
        }

        public function fscommand2NotSupported():Void{
              trace("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
        }

        private function isBelowWarningLevel():Void {
              if(Math.round(_currentLevel/_maxLevel*100) < _warningLevel){
                    trace("Status Level is LOW. ");
              } else {
                    trace("Status Level is FINE. ");
              }
        }

        public function get warningLevel():Number {
              return _warningLevel;
        }

        public function set warningLevel(level:Number):Void {
              warningLevel = level;
        }
}
```

An example of the `Signal` class is utilized in the `chapter3_determineSignalStrength.fla` file:




```
import com.wrox.chapter3.Signal;

var mySignal:Signal;
mySignal = new Signal();
mySignal.warningLevel = 10;
mySignal.determineSignalStrength();
```

codesSnippet chapter3_determiningSignalStrength.fla

In the code highlighted, you can see that the `fscommand2()` commands are assigned to a variable. The values returned from the `fscommand2()` will return a value based on the device manufacturer's implementation; for example, some devices have signal levels measured between 0 to 100, while others can have values between 0 to 8, or 0 to 7.



All commands will return –1 if the function is not supported by the device.

Monitoring Device Memory Allocation

The Flash Player memory allocation varies from device to device; `fscommand2()` also provides a way for you to determine the memory occupied by your application:

www.it-ebooks.info

- `GetTotalPlayerMemory` — To return the total player memory available for the player.
- `GetFreePlayerMemory` — To return the free memory available for the player.

Listing 3-14 gives an example usage of `fscommand2()` to provide the available memory in an application.



LISTING 3-14: The Memory class

```
class com.wrox.chapter3.Memory {

      public static var FSCOMMAND2_NOT_SUPPORTED:Number = -1;

      private var _currentLevel:Number;
      private var _maxLevel:Number;
      private var _warningLevel:Number;

      public function Memory(){
            _currentLevel = fscommand2("GetFreePlayerMemory");
            _maxLevel = fscommand2("GetTotalPlayerMemory");
            _warningLevel = 10;
      }

      public function determineAvailableMemory():Void {
            switch(_currentLevel){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  default:
                        trace("_currentLevel = " + _currentLevel);
                  break;
            }

            switch(_maxLevel){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  default:
                        trace("_maxLevel = " + _maxLevel);
                  break;
            }

            isBelowWarningLevel();
      }

      public function fscommand2NotSupported():Void{
            trace("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
      }

      private function isBelowWarningLevel():Void {
            if(Math.round(_currentLevel/_maxLevel*100) < _warningLevel){
                  trace("Memory Availability is LOW. ");
```

continues

www.it-ebooks.info

LISTING 3-14 *(continued)*

```
            } else {
                  trace("Memory Availability is FINE. ");
            }
      }

      public function get warningLevel():Number {
            return _warningLevel;
      }

      public function set warningLevel(level:Number):Void {
            warningLevel = level;
      }

}
```

code snippet Memory.as

The following snippet shows the code required to run the `Memory` class in your `.fla` file.


Available for download on Wrox.com

```
import com.wrox.chapter3.Memory;

var myMemory:Memory;
myMemory = new Memory();
myMemory.warningLevel = 10;
myMemory.determineAvailableMemory();
```

code snippet chapter3_determiningAvailableMemory.fla

One usage of the memory allocation is to instruct the user that they may be close to utilizing the memory required for maximum performance of an application. When you run the `chapter3_determiningAvailableMemory.fla` file in Device Central, have a go at changing the memory level of the device.

Edit the Static and Dynamic Heap values, so that the Static Heap is set to 300 and the Dynamic Heap is set to 0, as shown in Figure 3-4. The available memory should be "fine."

Figure 3-5 shows the Memory panel when the Static Heap has been set to 150. This is getting close to the limit.



FIGURE 3-4

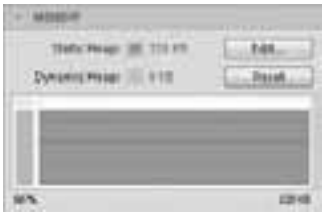


FIGURE 3-5

www.it-ebooks.info

After you change the heap values and run the code, have a look at the trace statements in the Output window of Device Central. When you set the Static Heap as low as it can go, you'll get the Flash Lite Error message "Out Of Memory Error!," which is shown in Figure 3-6.

Handling Network Requests and Connections

For applications that require use of data, you can keep the user informed of network requests and connection statuses with the following `fscommand2()` call:

`GetNetworkStatus` — To retrieve the current network status.

Listing 3-15 gives an example usage of `fscommand2()` to provide the available memory in an application, declaring each of the static variable values returned by the call for `GetNetworkStatus`.



FIGURE 3-6

Available for download on Wrox.com

LISTING 3-15: The NetworkStatus class

```
class com.wrox.chapter3.NetworkStatus {

      public static var FSCOMMAND2_NOT_SUPPORTED:Number = -1;
      public static var NOT_REGISTERED:Number = 0;
      public static var HOME_NETWORK:Number = 1;
      public static var EXTENDED_HOME_NETWORK:Number = 2;
      public static var ROAMING:Number = 3;

      private var _status:Number;

      public function NetworkStatus(){
           _status = fscommand2("GetNetworkStatus");
      }

      public function updateStatus(){
            switch(_status){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  case NOT_REGISTERED:
                        trace("No network is registered");
                  break;
                  case HOME_NETWORK:
                        trace("Network is on home network");
                  break;
```

continues

www.it-ebooks.info

LISTING 3-15 *(continued)*

```
                case EXTENDED_HOME_NETWORK:
                      trace("Network is on extended home network");
                break;
                case ROAMING:
                      trace("Network is on roaming");
                break;
          }
    }

    public function fscommand2NotSupported():Void {
          trace("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
    }
}
```

code snippet NetworkStatus.as

The following snippet shows the code required to use the `NetworkStatus` class in the `.fla` file.


Available for download on Wrox.com

```
import com.wrox.chapter3.NetworkStatus;

var networkStatus:NetworkStatus;
networkStatus = new NetworkStatus();
networkStatus.updateStatus();
```

code snippet chapter3_determiningNetworkStatus.fla

Unfortunately, the `fscommand2()` method isn't asynchronous, so you would need to create wrappers around each function so that you can monitor Network connection and request process:

- `GetNetworkRequestStatus` — To return the network request status.
- `GetNetworkConnectStatus` — To return the network connection status.

Listing 3-16 shows the `NetworkRequest` class, which determines the current status of a network request and handles all the possible returned values referenced in each case statement.


Available for download on Wrox.com

LISTING 3-16: The NetworkRequest class

```
class com.wrox.chapter3.NetworkRequest {

      public var FSCOMMAND2_NOT_SUPPORTED:Number = -1;
      public var PENDING_CONNECTION_TO_SERVER_MADE:Number = 0;
      public var PENDING_CONNECTION_BEING_ESTABLISHED:Number = 1;
      public var PENDING_CONNECTION_NOT_ESTABLISHED:Number = 2;
      public var PENDING_CONNECTION_ESTABLISHED_RESOLVING_SERVER_NAME:Number = 3;
      public var FAILED_NETWORK_ERROR:Number = 4;
      public var REQUEST_FAILED_CONNECTING:Number = 5;
      public var REQUEST_FAILED_SERVER_RETURNED_ERROR:Number = 6;
      public var REQUEST_FAILED_DNS:Number = 7;
```

www.it-ebooks.info

```
public var REQUEST_SUCCESS:Number = 8;
public var REQUEST_FAILED_TIMEOUT:Number = 9;
public var REQUEST_NOT_BEEN_MADE:Number = 10;

private var _status:Number;

public function NetworkRequest(){
      _status = fscommand2("GetNetworkRequestStatus");
}

public function updateStatus():Void {
      switch(_status){
            case FSCOMMAND2_NOT_SUPPORTED:
                  fscommand2NotSupported();
            break;
            case PENDING_CONNECTION_TO_SERVER_MADE:
                  trace("Pending request.");
                  trace("Network connection has been established.");
                  trace("Server host name has been resolved.");
                  trace("Connection to the server has been made.");
            break;
            case PENDING_CONNECTION_BEING_ESTABLISHED:
                  trace("Pending request.");
                  trace("Network connection is being established.");
            break;
            case PENDING_CONNECTION_NOT_ESTABLISHED:
                  trace("Pending request.");
                  trace("Network connection has not yet been established.");
            break;
            case PENDING_CONNECTION_ESTABLISHED_RESOLVING_SERVER_NAME:
                  trace("Pending request.");
                  trace("Network connection established.");
                  trace("Server host name is being resolved.");
            break;
            case FAILED_NETWORK_ERROR:
                  trace("Network error and the request has failed.");
            break;
            case REQUEST_FAILED_CONNECTING:
                  trace("Connecting to server and the request has failed.");
            break;
            case REQUEST_FAILED_SERVER_RETURNED_ERROR:
                  trace("The server has returned an error.");
            break;
            case REQUEST_FAILED_DNS:
                  trace("DNS server issue and the request has failed.");
            break;
            case REQUEST_SUCCESS:
                  trace("The request is a success.");
            break;
            case REQUEST_FAILED_TIMEOUT:
                  trace("Timeout and request has failed.");
            break;
```

continues

www.it-ebooks.info

LISTING 3-16 *(continued)*

```
                case REQUEST_NOT_BEEN_MADE:
                      trace("The request has not yet been made.");
                break;
          }
    }

    public function fscommand2NotSupported():Void {
           trace("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
    }
}
```

Lastly, Listing 3-17 shows a `NetworkConnection` class, which determines the current status of a network connection and handles all the possible values returned by the `fscommand2()` call, which are referenced in each case statement.



LISTING 3-17: The NetworkConnection class

```
class com.wrox.chapter3.NetworkConnection {

    public static var FSCOMMAND2_NOT_SUPPORTED:Number = -1;
    public static var CONNECTED:Number = 0;
    public static var TRYING:Number = 1;
    public static var NOT_CONNECTED:Number = 2;
    public static var SUSPENDED:Number = 3;
    public static var INDETERMINABLE:Number = 4;

    private var _status:Number;

    public function NetworkConnection(){
        _status = fscommand2("GetNetworkConnectStatus");
    }

    public function updateStatus():Void {

        switch(_status){
            case FSCOMMAND2_NOT_SUPPORTED:
                fscommand2NotSupported();
            break;
            case CONNECTED:
                trace("Connected");
            break;
            case TRYING:
                trace("Trying To Connect");
            break;
            case NOT_CONNECTED:
                trace("Not Connected");
            break;
            case SUSPENDED:
                trace("Suspended");
            break;
```

www.it-ebooks.info

```
                case INDETERMINABLE:
                      trace("Indeterminable");
                break;
            }
        }

        public function fscommand2NotSupported():Void {
              trace("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
        }
    }
```

If you are more familiar with developing for the Flash Lite 1.1 player, the support of AS2 from Flash Lite 2.0 onward means that parts of the AS1 code base has been deprecated. This includes 15 of the original `fscommand2()` calls and arguments, which have now been replaced as preferred methods or properties on core classes. For example, the `System.capabilities.language` property should be used instead of the `GetLanguage` command, and the `Date` object should be used instead of the commands that were used to retrieve time-based information from the device.

Other Scenarios for Using fscommand2()

The following list of the scenarios and the `fscommand2()` are required for achieving the task:

- Monitoring the device volume
  - `GetVolumeLevel` — To return the current volume level.
  - `GetMaxVolumeLevel` — To return the maximum volume level available.
- Managing device vibration
  - `StartVibrate` — To start vibration.
  - `StopVibrate` — To stop vibration.

For the `StartVibrate` call, you have to specify three additional parameters to the `fscommand2()` call. The second parameter is the length of time in milliseconds that vibration is on for, the third parameter is the length of time in milliseconds that the vibration is off for, and the fourth parameter is the repeat count. The time values can have a maximum of 5 seconds assigned, while the repeat count can be done a maximum of three times.

- Exit an application:
  - `Quit`
- Maximize the view:
  - `FullScreen`
- Set the text input type for the device hosting the Flash Lite player:
  - `SetInputTextType`

The `SetInputTextType` call requires you to specify two additional parameters to the `fscommand2()` call. The second parameter is a reference to the `variable` property of the `TextField`

www.it-ebooks.info

within the Flash Lite application, and also the type of text you want the input type to be restricted to. Your options for the third parameter include: `Numeric`, `Alpha`, `Alphanumeric`, `Latin`, `NonLatin`, and `NoRestriction`.

- Extend the duration of the backlight for a device:
  - `ExtendBackLightDuration`

The `ExtendBackLightDuration` call requires you to specify the duration time, in seconds as the second parameter.

- Changing the default color for the focus rectangle:
  - `setFocusRectColor`

Using the `setFocusRectColor` requires you to supply three additional parameters for "red," "green," and "blue."

In the section *An OOP Example for Mobile*, you learned how to create a generic `Mobile` class, which then could be extended to create other types of mobile devices, in particular the Sony Ericsson Aino and the Nokia E71. Of those base properties mentioned for the `Mobile` class, there are several that can be retrieved from the application using `fscommand2()` commands, including:

- `GetDeviceID` — To return the unique identifier for the device running the Flash Lite application, such as the IMEI or Serial Number.
- `GetPlatform` — To return the current platform for the device running the Flash Lite application.
- `GetDevice` — To return the current device running the Flash Lite application.

For each of these command calls, you need to supply an additional parameter: a string value representing the variable to store the value being retrieved.

While we've covered a great deal in this section, for a comprehensive list and more information on the `fscommand2()` commands, visit the Adobe Live Docs Web site at `http://livedocs.adobe.com/flashlite/3.0/docs/help.html?content=00005027.html#206861`.

USING DEVICE CAPABILITIES

As you've probably gathered by now, it's important to consider which environment your mobile device application will run in, since not all devices running Flash Lite will have the exact same features. If you remember, the Sony Ericsson Aino and the Nokia E71 have different ways for users to interact with software applications. The E71 has a QWERTY keypad, similar to the one found on most laptops and PCs, making it easier to write an email, for example. The Aino has both Touch UI and *traditional* mobile keypad capabilities, allowing a user to choose between Touch UI, by pressing the screen of the device, or using the keys for navigation. Later you'll cover using keys and touch interaction.

www.it-ebooks.info

In this section you will learn about the `System.capabilities` class, which can be used for accessing more information about your application's environment.

System.capabilities

Use the `System.capabilities` object to determine the capabilities of a device running the Flash Lite player. This class has numerous static `read-only` properties that can provide you with the data to help you adapt your application to the targeted system, based on the values retrieved by `System.capabilities`.

Consider the need to build an application that requires playing an MP3 or Video, or an application that needs to send MMS. Only, you find that the device doesn't have the facility to playback the media in the way you need and the device is only able to send SMS. "Duh!"

The `System.capabilities` properties can be grouped into the following categories:

- Information about the system
- External data support
- Key and device interaction
- Media capabilities
- Communication features

You'll take a look at each one of these in turn next. There are also files accompanying each section, which you can run to view each of the properties listed.

Information about the System

You can use the system details in an application to specify information about the device's hardware and software. Three properties of the `System.capabilities` object in particular can provide this:

- `System.capabilities.os` — Returns the operating system.
- `System.capabilities.version` — Returns the Flash Lite player version.
- `System.capabilities.language` — Returns the device's language.

You can use these properties for various scenarios. For instance, knowing about a device's default language in the application could provide a useful aid. In the event that the end user changes the default language on the device, the application could adapt to this automatically, by detecting the language property.

If you run the `chapter3_SystemInfoSupport.fla` file, you will see each of the system information details supported by a device in Device Central. The code in the `.fla` file simply contains an Input text field set on the stage, with its "variable" setting set to `capabilities`. You'll find that, on the first frame of the Timeline, the AS sets the movie to `FullScreen` and then assigns each of the properties listed to the `capabilities` variable (see Listing 3-18). The code is similar for the subsequent sections on `System.capabilities`.

www.it-ebooks.info



LISTING 3-18: The System.capabilities object properties for System Information

```
fscommand2("FullScreen", true);

capabilities = "SYSTEM INFORMATION: " + newline + newline;

capabilities += "os = " + System.capabilities.os + " | ";
capabilities += "version = " + System.capabilities.version + " | ";
capabilities += "language = " + System.capabilities.language + " | ";
```

External Data Support

To determine whether you can use particular functions and methods in AS, there are numerous properties of the `System.capabilities` object that can return particular information relating to external data:

- `System.capabilities.hasDataLoading` — Returns whether the Flash Lite player can dynamically load additional data through several calls, including: `loadMovie()`, `loadMovieNum()`, `loadVariables()`, `loadVariablesNum()`, `XML.parseXML()`, `Sound.loadSound()`, `MovieClip.loadVariables()`, `MovieClip.loadMovie()`, `MovieClipLoader.loadClip()`, `LoadVars.load()`, and `LoadVars.sendAndLoad()`. The majority of these methods are covered throughout the book.
- `System.capabilities.hasSharedObjects` — Returns whether the content can access Flash Lite shared objects.
- `System.capabilities.hasXMLSocket` — Returns whether the host application supports XML sockets.

If you run the `chapter3_ExternalDataSupport.fla` file, you will see each of the data features supported by a particular device in Device Central.

Key and Device Interaction

There are six properties in `System.capabilities` that can help in determining and making the right choice of key input for device applications:

- `System.capabilities.hasMappableSoftKeys` — Returns whether or not a user can set soft key values and handle events from those soft keys.
- `System.capabilities.softKeyCount` — Returns a number specifying the total number of soft keys that the platform supports.
- `System.capabilities.has4WayKeyAS` — Returns whether the player can execute ActionScript attached to Key Event handlers associated with the right, left, up, and down keys.
- `System.capabilities.hasQWERTYKeyboard` — Returns whether or not ActionScript can be attached to all keys found on a standard QWERTY keyboard and the Backspace key.

www.it-ebooks.info

- `System.capabilities.hasMouse` — Returns whether or not the platform supports mouse.
- `System.capabilities.hasStylus` — Returns whether or not the player can send stylus-related events.

When you run the `chapter3_KeySupport.fla` file, you will see each of the key and interactivity features supported by a device in Device Central.



There are a few terms here which relate to the sections Events and Event Listeners and Keys, Buttons, and Touch Input, which you'll cover shortly.

Media Capabilities

The media capabilities of the device can also be detected from Flash Lite content through the `System.capabilities` object:

- `System.capabilities.audioMIMETypes` — Returns an array of MIME types that the device's audio codecs support, which can be used by the `Sound` object.
- `System.capabilities.hasAudio` — Returns whether or not the device supports audio.
- `System.capabilities.hasCMIDI` — Returns whether or not the platform supports CMIDI sound.
- `System.capabilities.hasCompoundSound` — Returns if the player can process compound sound data.
- `System.capabilities.hasEmbeddedVideo` — Returns whether or not the player can process embedded video.
- `System.capabilities.hasMFI` — Returns whether the player can play sound data in the Melody Format (MFi) audio format.
- `System.capabilities.hasMIDI` — Returns whether the player can play sound data in the MIDI audio format.
- `System.capabilities.hasSMAF` — Returns whether the player can play sound data in the Synthetic [music] Mobile Application Format (SMAF).
- `System.capabilities.hasStreamingAudio` — Returns whether or not the device supports streaming audio.
- `System.capabilities.hasMP3` — Returns whether or not the device supports MP3.
- `System.capabilities.hasStreamingVideo` — Returns whether or not the player can stream video.
- `System.capabilities.imageMIMETypes` — Returns an array of MIME types that the device's image codecs support, which can be used by `loadMovie()`.

www.it-ebooks.info

- `System.capabilities.MIMETypes` — Returns an array of all the MIME types that the `Sound` and `Video` objects support, which can be used by `loadMovie()`.
- `System.capabilities.videoMIMETypes` — Returns an array of MIME types that the device's video codecs support, which can be used by the `Video` object.

If you run the `chapter3_MediaSupport.fla` file, you will see each of the media capabilities listed supported by a device in Device Central. You'll get to cover the media capabilities of the Flash Lite Player in more detail in Chapter 7.

Communication Features

If you need your application to use email, SMS, or MMS, you can use the `System.capabilities` object to indicate whether these features are supported:

- `System.capabilities.hasEmail` — Returns whether or not the player can send SMS.
- `System.capabilities.hasSMS` — Returns whether or not the player can send short message service (SMS) messages using `getURL()`.
- `System.capabilities.hasMMS` — Returns whether or not the player can send MMS.

When you run the `chapter3_CommunicationSupport.fla` file, you will see each of the communication features listed supported by a device in Device Central.

EVENTS AND EVENT LISTENERS

Consider what happens on a mobile device when a key is pressed. Behind the scenes in a Flash Lite application, there is a small amount of AS code that activates a transaction between the broadcast and receiving of events by an object.

Events are the communication mechanism by which classes talk with each other in AS. Events are broadcast by one object and then another object, which listens for an event and invokes a handler function, which is implemented as a response to the event. In the last section you learned how you can detect the key and interaction capabilities of a device. Before actually tackling the final section in this chapter, *Keys, Buttons, and Touch Input*, you'll take a look at events and event listeners. First, you'll take a look at the `EventDispatcher` class, which can be used in Flash Lite OOP to create custom classes that dispatch events.

Using the EventDispatcher Class

In order to broadcast an event, you must *initialize* your object by registering it with the `EventDispatcher` class. First, this requires you to import the `mx.events.EventsDispatcher` package into your class, and then declare the following three public variables with the data type `Function`:

- `addEventListener()`
- `dispatchEvent()`
- `removeEventListner()`

www.it-ebooks.info

We'll cover these methods shortly. In the constructor of the class from which you want to dispatch an event, you need to call the `EventsDispatcher.initialize()` method, which must be called to register and initialize your class. You supply the `this` keyword as a reference parameter for the object created. All this will be explained in more detail next.

Creating the Timer Class Example

In the following example, you will create two classes that utilizes AS2 event handling in Flash Lite:

- `Timer`
- `TimerListener`

First, create the `Timer` class. This has the task of counting down from 10 seconds to zero and then dispatches an event to say it has completed (see Listing 3-19).



LISTING 3-19: Creating the Timer class

```
import mx.events.EventDispatcher;

class com.wrox.chapter3.Timer {

      public var addEventListener:Function;
      public var dispatchEvent:Function;
      public var removeEventListener:Function;

      public function Timer(){
            EventDispatcher.initialize(this);
      }
}
```

The `Timer` class will need to utilize the internal Flash Player clock; for this you can use the intrinsic `setInterval()` to invoke a clock reference, and `clearInterval()` to remove it when we've finished using it.

setInterval() and clearInterval()

The value returned by the `setInterval()` function should be assigned to a variable, in order to be referenced later. The function takes two parameters: a reference to a named handler function, and the duration of the interval in milliseconds. The handler function is invoked when the interval value set for `setInterval()` expires; it then restarts. In order to prevent the handler from being called again, the `clearInterval()` function needs to be called, passing in one parameter, the reference variable to `setInterval()`.

Continuing with the `Timer` example, create a public method `init()` to kick start the countdown, then implement the private handler function `tickComplete()`, which traces out the current timer value, `tickCount`, to the output panel. The private variable `timerId` will store the value returned by `setInterval()`, and `duration` should be set to 1000 milliseconds, equal to one second (see Listing 3-20).

www.it-ebooks.info



LISTING 3-20: The init and tickComplete methods for Timer

```
import mx.utils.Delegate;
import mx.events.EventDispatcher;

class com.wrox.chapter3.Timer {

      public var addEventListener:Function;
      public var dispatchEvent:Function;
      public var removeEventListener:Function;

      private var duration:Number = 1000;
      private var tickCount:Number = 10;
      private var timerId:Number;

      public function Timer(){
            EventDispatcher.initialize(this);
      }

      public function init():Void {
            trace("init():: tickCount = " + tickCount);
            timerId = setInterval(Delegate.create(this, tickComplete), duration);
      }

      private function tickComplete():Void {
            tickCount—;
            trace("tickComplete():: tickCount = " + tickCount);
            if(tickCount == 0){
                  clearInterval(timerId);
            }
      }
}
```

When the `init()` method is called, the timer starts, and with every subsequent second that passes, the `tickCount` value decreases. This is because the `tickComplete()` method is called. When the `tickCount` variable reaches zero, it can then fire off an event to say that the timer is complete. To do this the `dispatchEvent()` method needs to be used.

dispatchEvent()

The `dispatchEvent()` method takes as a parameter an object that has to have two properties defined: the `target` property as an object and `type` property as a string. Listing 3-24 shows the `tickComplete()` handler method, which has been modified to include the `dispatchEvent()` and its reference `target` property, which is set to `this`, the scope of the `Timer` class itself. The `type` property is also set to the event signature `onTimerCompleted` (see Listing 3-21).



LISTING 3-21: Using dispatchEvent in the tickComplete method for Timer

```
private function tickComplete():Void {
      tickCount—;
```

www.it-ebooks.info

```
        if(tickCount == 0){
              clearInterval(completeInterval);
              dispatchEvent({target:this, type:"onTimerCompleted"});
        }
        trace("init():: tickCount = " + tickCount);
  }
```

addEventListener() and removeEventListener()

Now consider how an object is going to receive the dispatched event fired from the `tickComplete()` method of the `Timer` class.

Listing 3-22 shows how this can be achieved through a class called `TimerListener`. Here, the `Timer` class is imported, along with the `Delegate` class. An instance of the `Timer` called `myTimer` is created when `TimerListener` is instantiated, and before the `init()` method is called, the `addEventListener()` method of `myTimer` is also called.



LISTING 3-22: The TimerListener class

```
import com.wrox.chapter3.Timer;
import mx.utils.Delegate;

class com.wrox.chapter3.TimerListener {

      private var myTimer:Timer;

      public function TimerListener(){
            myTimer = new Timer();
            myTimer.addEventListener("onTimerCompleted", Delegate.create(this,
onComplete));
            myTimer.init();
      }

      private function onComplete():Void {
            trace(" TimerListener:: onComplete() ");
            myTimer.removeEventListener("onTimerCompleted");
            myTimer.removeEventListener("onTimerCompleted", Delegate.create(this,
onComplete));
      }

}
```

With `addEventListener()` you specify two parameters, the event `type` sent by the `Timer` object, which is `onTimerCompleted` in this example, and the event handler function `onComplete()`, which is assigned to the second parameter. So when the `onTimerCompleted` event is retrieved, the output panel will display the text `TimerListener:: onComplete()`. The `removeEventListener()` is called on the `myTimer` object to remove the event listener once the timer has counted down to zero (see Listing 3-22).

www.it-ebooks.info

There are just two lines to run the example:




```
import com.wrox.chapter3.TimerListener;
var myTimerListenerObj:TimerListener = new TimerListener();
```

Code Snippet chapter3_Timer.fla

You've now covered aspects of events and event listeners, which are a key part of coding in AS and Flash Lite development. Next, you'll cover user's key input, which is paramount to a fully functioning application.

KEYS, BUTTONS, AND TOUCH INPUT

The keypad is one of the central features for the majority of mobile devices in the market. For your Flash Lite mobile applications, "key input" implementation is of the upmost importance for successful user interaction on the mobile device for controlling your application. Throughout this book you'll cover numerous examples of navigating through an application. Here, you'll cover using the `Key` class, using Buttons, and handling Touch Input.

Using the Key Class

In AS2 you use the intrinsic `Key` class to handle the interactions from the user. The methods and properties of the `Key` class are all static, so you don't use a constructor to create a `Key` object.

Key.addListener()

In order to capture a user's key input, you can use `Key.addListener()`. The method takes a "listener" object as an argument, which it registers and saves a reference to in the `Key._listeners` property. The `Key` object will then monitor the key input from the device and notify the key listener objects stored in the `_listeners` property, invoking the event handlers `Key.onKeyDown()` and `Key.onKeyUp()`, which must be defined for the listener object.



LISTING 3-23: Using the Key.addListener method

```
import mx.utils.Delegate;

var keyListener:Object = {};
keyListener.onKeyDown = Delegate.create(this, onKeyDown);
keyListener.onKeyUp = Delegate.create(this, onKeyUp);

Key.addListener(keyListener);

function onKeyDown() {
      trace(" A key was pressed.");
}

function onKeyUp() {
      trace(" The key was released.");
}
```

www.it-ebooks.info

onKeyDown() and onKeyUp()

The `Key.onKeyDown()` event handler is invoked when a user presses *down* on a key, whereas the `Key.onKeyUp()` is mapped to when a user releases a key on the device. Among the methods and properties of the `Key` object, `Key.getCode()` allows you to retrieve the last key pressed on the device. This has the benefit of determining what key the user pressed and how you can respond to it.

Depending on what key is pressed, the `Key.getCode()` method returns a value of type number or string. The `Key.getCode()` is a very useful method. You can cross-reference this value with the static key code properties of the `Key` object, such as `Key.ENTER`. The method allows you to capture specific keys at runtime, and then invoke your own implementation based on what key the user pressed.

Table 3-2 displays some of the static key code properties for the `Key` class along with their corresponding values, which are returned when the `Key.getCode()` method is called.

TABLE 3-2: Static properties for the key class with corresponding key codes

PROPERTY	KEY CODE
BACKSPACE	8
CAPSLOCK	20
CONTROL	17
DELETEKEY	46
DOWN	40
END	35
ENTER	13
ESCAPE	27
HOME	36
INSERT	35
LEFT	37
PGDN	34
PGUP	33
RIGHT	39
SHIFT	16
SPACE	32
TAB	9
UP	38

www.it-ebooks.info

Listing 3-24 shows an example of the `onKeyDown()` event handler function modified to capture when the Enter key is pressed on a device and when other keys are also pressed. The key value is traced in the Output window.



LISTING 3-24: Using the Key.getCode method in the Key.onKeyDown event handler

```
import mx.utils.Delegate;

var keyListener:Object = {};
keyListener.onKeyDown = Delegate.create(this, onKeyDown);
keyListener.onKeyUp = Delegate.create(this, onKeyUp);

Key.addListener(keyListener);

function onKeyDown() {
      switch(Key.getCode()) {
            case Key.ENTER:
                  trace(" The Enter key was pressed. " + Key.getCode());
            break;
            default:
                  trace(" Another key was pressed. " + Key.getCode());
            break;
      }
}

function onKeyUp() {
      trace(" The key was released. ");
}
```

Key.removeListener()

Use the `Key.removeListener()` method in instances where you no longer need or want to capture key input, or when you need to remove an object, so it no longer has its `onKeyDown()` or `onKeyUp()` functions invoked. As with the `Key.addListener()` method you supply a reference to the key listener object as a parameter.

Listing 3-25 shows an example where the `onKeyUp()` event handler function has been modified to capture when the Enter key is released on a device. When the Enter key is pressed, the key listener object is removed from the `Key` class via the `Key.removeListener()` method, preventing any further key events from being broadcast via the listener object.



LISTING 3-25: Using the Key.removeListener method in the Key.onKeyUp event handler

```
function onKeyDown() {
      switch(Key.getCode()) {
            case Key.ENTER:
                  trace(" The Enter key was pressed. " + Key.getCode());
            break;
            default:
                  trace(" Another key was pressed. " + Key.getCode());
```

www.it-ebooks.info

```
            break;
      }
}

function onKeyUp() {
     switch (Key.getCode()) {
            case Key.ENTER:
                  trace(" The Enter key was pressed. " + Key.getCode());
                  trace(" Key.removeListener() is called. ");
                  trace(" So no more trace output. " );

                  Key.removeListener( keyListener );
            break;
            default:
                  trace(" Another key was released. " + Key.getCode());
                  trace(" Key.removeListener() was not called. ");
                  trace(" Hence continue the trace output. ");
            break;
     }
}
```

It is best practice to always remove the key listener object when you no longer need it.

Using the ExtendedKey Class

In Flash Lite the key code values returned by a device are universal, which means you can map keys to specific parts of your application. There are additional keys on devices with *traditional* keypads and QWERTY keyboards called "soft keys." There are usually soft-key labels displayed on the screen of a device for navigational purposes and to indicate each soft key's function.

The `ExtendedKey` class simply contains 12 static key code properties, which return the value for a soft key on a device. When soft keys are pressed, their key codes are returned and can be captured from the `Key.getCode()` method. For instance when the left soft key on the device is pressed, the key code `ExtendedKey.SOFT1` is returned. When the right soft key is pressed, the `ExtendedKey.SOFT2` key code is returned (see Listing 3-26).



LISTING 3-26: Capturing the ExtendedKey presses in the Key.onKeyDown and Key.onKeyUp event handlers

```
function onKeyDown() {
      switch(Key.getCode()) {
            case Key.ENTER:
                  trace(" The Enter key was pressed. " + Key.getCode());
            break;
            case ExtendedKey.SOFT1:
                  trace(" The LEFT soft key was pressed. " + Key.getCode());
            break;
            case ExtendedKey.SOFT2:
                  trace(" The RIGHT soft key was pressed. " + Key.getCode());
            break;
```

continues

www.it-ebooks.info

LISTING 3-26 *(continued)*

```
            default:
                  trace(" Another key was pressed. " + Key.getCode());
            break;
      }
}

function onKeyUp() {
     switch (Key.getCode()) {
            case Key.ENTER:
                  trace(" The Enter key was pressed. " + Key.getCode());
                  trace(" Key.removeListener() is called. ");
                  trace(" So no more trace output. " );

                  Key.removeListener( keyListener );
            break;
            case ExtendedKey.SOFT1:
                  trace(" The LEFT soft key was pressed. " + Key.getCode());
            break;
            case ExtendedKey.SOFT2:
                  trace(" The RIGHT soft key was pressed. " + Key.getCode());
            break;
            default:
                  trace(" Another key was released. " + Key.getCode());
                  trace(" Key.removeListener() was not called. ");
                  trace(" Hence continue the trace output. ");
            break;
      }
}
```

The soft key labels mentioned can be set via the `fscommand2()` command `SetSoftKeys`, which takes a number of additional parameters to specify the soft key labels; the number of parameters is based on the number of soft keys available on the device. Remember, `System.capabilities.softKeyCount` can return this property, and then you could define the labels based on this.

Handling Touch Interaction

With the absence of keypads in many new touch screen devices, it may not be appropriate to implement key listeners in all your applications. Of course, this depends on the device you are targeting, but for touch screens you can use Buttons or MovieClips for interaction. You'll find some devices have both keypad and touch screen capabilities, which gives you the option of implementing handlers for both Key, Button, and MovieClip events.

You can use the `System.capabilities.hasStylus` property as an indication as to whether or not a device has the ability to accept user input via the screen.

Using Buttons

If you are developing for devices with touch screen capability, you simply create an instance of a `Button` or `MovieClip` element and then assign functions to event handlers when specific events are

www.it-ebooks.info

invoked. These event handlers are particularly useful for targeting touch-screen-based devices, and include the following:

- `onDragOut()` — Used when a user touches an element in the application and then drags a finger outside the boundary of that element.
- `onDragOver()` — Used when a user has first invoked the `onPress()` event, then the `onDragOut()` event, and drags a finger back over the element.
- `onPress()` — Used when a user touches an element in the application. Here the element is said to "gain focus."
- `onRelease()` — Used when a user releases his or her touch from the element.
- `onReleaseOutside()` — Used when a user releases his or her touch outside of its viewable area.
- `onRollOut()` — When a user has first invoked the `onPress()` event, then the `onRelease()` event, and then touches the screen outside the boundary for the element.
- `onRollOver()` — When a user drags a finger over the element.

The following code snippet can be run in the first frame of the Timeline of a `.fla` file with a MovieClip added to the stage with the instance name of "square". The code outputs each of the events mentioned when run on a touch device in Device Central, such as the Nokia 5800 XpressMusic.




```
import mx.utils.Delegate;

square.onDragOut = Delegate.create(this, onDragOut);
square.onDragOver = Delegate.create(this, onDragOver);
square.onPress = Delegate.create(this, onPress);
square.onRelease = Delegate.create(this, onRelease);
square.onReleaseOutside = Delegate.create(this, onReleaseOutside);
square.onRollOut = Delegate.create(this, onRollOut);
square.onRollOver = Delegate.create(this, onRollOver);

function onDragOut() {
       trace(" onDragOut. ");
}

function onDragOver() {
       trace(" onDragOver. ");
}

function onPress() {
       trace(" onPress. ");
}

function onRelease() {
       trace(" onRelease. ");
}

function onReleaseOutside() {
```

www.it-ebooks.info

```
        trace(" onReleaseOutside. ");
}

function onRollOut() {
        trace(" onRollOut. ");
}

function onRollOver() {
        trace(" onRollOver. ");
}
```

code snippet chapter3_Touch.fla

SUMMARY

In this chapter, you've covered a lot of programming.

You learned about object-oriented programming principles in the context of Flash Lite development. You then took a look at some of the key considerations for developing Flash Lite applications and covered various scenarios for communicating with a mobile device using the `fscommand2()`.

While using the `System.capabilities` object, you also learned how to retrieve the properties and functional capabilities of a device running Flash Lite. This gave you an insight into how mobile devices can have different properties and, in particular, different modes of user input. You then also learned about more core features of AS in events and also keys, buttons, and Touch Input.

In the next chapter, you'll take a look at more of the visual elements that you can use to build Flash Lite applications.

www.it-ebooks.info

4

UI Components

WHAT'S IN THIS CHAPTER?

- Sony Ericsson component libraries
- Handling component events
- Setting UI component styles

Compared to some other mobile technologies, you have a lot more creative freedom in Flash to design every aspect of your mobile application. However, when you are seemingly presented with a "blank canvas" at the start of your creation, developing the building blocks for your application, without any sort of guidance, can at first appear to be quite daunting.

Developing the *user interface* for a mobile application is made a whole lot easier when you have an array of ready-made component elements to use. Enter User Interface (UI) Components.

One of the advantages in using UI components is that you will undoubtedly spend less time thinking about the intricacies of developing elements such as list menus, image sliders or status bars and give yourself more development time to spend on the overall functionality and higher-level logic of your mobile application, which is one of the main aims of this book. In addition, implementing recognizable UI elements gives end users a much more familiar and consistent user experience.

In this chapter you explore one the most comprehensive UI component libraries developed for Flash Lite application development, created by Sony Ericsson.

You'll learn how to programmatically integrate a range of components that you can use as part of your applications, and also learn how to define the UI component styles, implement their events, call methods, and set properties that change aspects of the UI component's behavior.

So the aim of this chapter is to give you a head start in producing high-quality user interfaces.

www.it-ebooks.info

SONY ERICSSON AND FORUM NOKIA COMPONENT LIBRARIES

Both Forum Nokia and Sony Ericsson released UI components to address in part the absence of a dedicated UI component framework for Flash Lite, but also to drive consistency in design for Flash Lite applications across their handset range, resulting in some extremely handy assets for building applications.

The majority of UI components are made up of a combination of the `MovieClip`, `TextField`, and `Button` assets, all of which you should already be very familiar with.

While the Flash IDE comes with UI components for developing Flash applications pre-installed, it's widely accepted that these components are not fully optimized for Flash Lite and contain overhead that can actually reduce the performance of an application over time.

The following list details the Sony Ericsson UI components that you'll cover in this chapter:

- Buttons
- Check boxes
- Radio buttons
- Lists
- Modal dialogues
- Soft keys
- Indicators
- Sliders
- Status bar
- Scrollable areas
- Title
- Notifications

To follow the sample listings in this chapter, you will need to install the `Sony_Ericsson_UI_Components_2.0.mxp` via the Adobe Extension Manager. Once installed, the Sony Ericsson UI components will be ready for use and can be accessed from the Components Panel in the Flash CS4.

Figure 4-1 shows all the version 2.0 Sony Ericsson UI components in the Components Panel after installing the MXP.



FIGURE 4-1

Each sample implementation of the Sony Ericsson UI components for the remainder of this chapter can be found in the accompanying chapter4.zip file. There's a fair amount to cover here, so each

www.it-ebooks.info

code listing in the chapter can be run by simply adding the code to the first frame of the timeline and importing the necessary components referenced into the Library Panel.

Before moving onto these components let's just remind ourselves of one of the key components in Flash, the `TextField`.

USING TEXT

The `TextField` component is one of the simplest UI components to get your head round.

There are three types of `TextField` you can use in Flash:

- **Static text** — A read-only text field that allows automatic word wrapping, multiline text, and string formatting, including left, right, center, and justified paragraph alignment. Static text is created at development time.
- **Input text** — An editable text entry field that supports multiline text, string formatting, and word wrapping. You can also dynamically create input text at application runtime.
- **Dynamic text** — A writable text field that can be altered at runtime through variables.

You can create input and dynamic text to render HTML and also embed fonts; however, the latter is not recommended.

Here is a simple example of creating a `TextField` using AS (ActionScript); the script is placed on the first frame of the timeline:




```
this.createTextField("myTextField", this.getNextHighestDepth(), 0, 0, 240, 20);
myTextField.textColor = 0xFF0000;
myTextField.text = "Hello World!";
```

code snippet chapter4_TextField.fla

Next, let's take a look at buttons.

USING BUTTONS

Buttons are also very simple components to grasp. The basic button should consist of three visual states and allow the user to interact with the interface of the application.

As with the `TextField` component, you can create your own buttons with ease by selecting the Insert ➪ New Symbol ➪ Type ➪ Button method options when authoring in Flash CS4.

Both Nokia and Sony Ericsson have created out-of-the-box button UI components that make it easier to style buttons and integrate them into an application, and both are neatly designed controls that give you a *state-ready* component, which is ready to respond to user interaction.

www.it-ebooks.info

Here is a list of features that highlight the benefits of using the Button UI components:

- **State ready** — There's no need for you to create "up," "down," and "over"; the states of the button are already implemented.
- **Ability to set the text label** — There's no need to attach text fields in each state.
- **Position the text alignment** — You can position the text left, right, or center without having to change the x position.
- **Option to set an icon** — You can include a graphic alongside the text or have the icon displayed on its own.

These features are common across both the Nokia and Sony Ericsson component sets. The Nokia version of the Button component allows you to provide your own skin which you reference from the Library. For more information on the Nokia components, I recommend visiting the online "Flash Lite Components" page at `http://www.forum.nokia.com/info/sw.nokia.com/id/430ed7e3-dae8-481a-a3d7-e00ff8c1624c/Flash_Lite_Components.html`.

UI Component States

The Sony Ericsson version of a button, called "push button," has five states:

- **Enabled** — The user hasn't interacted with the button, and the button is available for highlighting and selection. The button is deemed enabled but "out of focus"; this state also represents the "up."
- **Highlighted** — The user has moved the focus to the button, and the button is available for selection or *pressing*. The button is deemed to be enabled and "in focus"; this state also represents the "over" state.
- **Pressed** — The user has selected and pressed the button. Here the button is deemed to be pressed and selected; this state also represents the "down" state.
- **Disabled** — The button is out of focus and is not available to be selected or pressed.
- **Disabled and Highlighted** — The user has moved the focus to a button that is in a disabled state.



You'll notice that the majority of component properties covered in this chapter share a consistency in naming for the states, which should make it easier for you to grasp.

By default the button is *enabled*, meaning that the button is active and a user can interact with it.

Using the Component Inspector

While we are covering the coding side of using the Sony Ericsson UI components, you should be aware that the components have also been created with drag and drop capability in mind.

www.it-ebooks.info

This means that you can simply drag the components from the Components Panel and drop them directly onto the stage and then change the default properties, including the text and styles of the component through the Component Inspector Panel.

To change the properties of the default properties of the button, for example, go through the following steps.

1. Open the Component Library. Select Window ➪ Components.
2. In the panel that opens, select and drag a push button component from the Sony Ericsson list to the stage.
3. Ensure that you have the push button component selected while on the stage.
4. Open the Component Inspector. Select Window ➪ Component Inspector.
5. Change the Background color value from #FAFAFA to #0066CC.

During this transition you should note that the background color of the push button changes while it is on the stage, giving you a live preview of changes in the Component Inspector.

Figure 4-2 shows the default settings for the push button in the Component Inspector Panel.

The example in Listing 4-1 demonstrates the full functionality of the button component, its methods, and its properties.

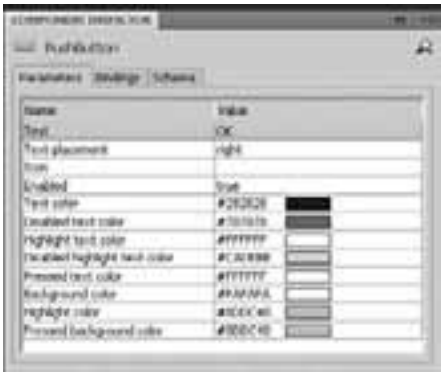


FIGURE 4-2



LISTING 4-1: Using the PushButton UI Component

```
addButton("btnOne", 10, 70);
addButton("btnTwo", 10, 150);

function addButton(btnName, x, y){

      this.attachMovie("PushButton", btnName, this.getNextHighestDepth());

      this[btnName]._x = x;
      this[btnName]._y = y;

      this[btnName]._labelText = btnName;

      this[btnName]._textColor = 0xFFFFFF;
      this[btnName]._backgroundColor = 0x000000;

      this[btnName]._disabledHighlightTextColor = 0xFFFFFF;
      this[btnName]._disabledTextColor = 0xFFFFFF;

      this[btnName]._highlightColor = 0xFFFFFF;
```

continues

www.it-ebooks.info

LISTING 4-1 *(continued)*

```
        this[btnName]._highlightTextColor = 0x000000;

        this[btnName]._pressedBackgroundColor = 0xFF0000;
        this[btnName]._pressedTextColor = 0xFFFFFF;

        this[btnName].onFocusIn = onFocusInHandler;
        this[btnName].onFocusOut = onFocusOutHandler;
        this[btnName].onSelect = onSelectHandler;

}

function onSelectHandler(){
        this._labelText = "I have been pressed";
}

function onFocusInHandler(){
        this._labelText = "I am highlighted";
}

function onFocusOutHandler(){
        this._labelText = "I am not highlighted";
}
```

The example shows the addition of two buttons, "btnOne" and "btnTwo", to the stage using the `addButton()` method, which takes three parameters, an instance name called `btnName`, and two coordinates, `x` and `y`, to position the newly created button relative to the stage:




```
addButton("btnOne", 10, 70);
addButton("btnTwo", 10, 150);
```

code snippet chapter4_PushButton.fla

The code requires that you have the push button component in the library. If you drag the component to the stage, it automatically adds this to the library; you then just need to delete it from the stage.

The example uses the Linkage Identifier reference name `PushButton` to draw the component to the stage via the `attachMovie()` method:




```
this.attachMovie("PushButton", btnName, this.getNextHighestDepth());
```

code snippet chapter4_PushButton.fla

The first parameter supplied to `attachMovie()` is the library linkage. The second parameter supplied is the `btnName`, a unique identifier for the button. The identifier value has to be unique for every new instance of button; likewise, the final parameter of `attachMovie()` is the depth, which also has to be unique. Using `getNextHighestDepth()` method ensures that a new depth is returned for the new instance.

The next few lines in the `addButton()` method handle the button's alignment on the stage.

www.it-ebooks.info




```
this[btnName]._x = x;
this[btnName]._y = y;
```

code snippet chapter4_PushButton.fla

The first button created in the example, with the instance name "btnOne", is aligned 10 pixels along the x axis and 70 pixels down the y axis of the stage. In the `addButton()` method, you will also see that the text for each push button is set through the `_labelText` property.

Both the text and button background color can be changed according to the state of the button. The enabled state can be styled by setting the `_backgroundColor` and `_textColor` properties. The highlighted state is styled through the `_highlightColor` and `_highlightTextColor` properties. The disabled state can be styled through the `_disabledHighlightTextColor` and `_disabledTextColor` properties. Finally, the pressed state can be styled through the `_pressedBackgroundColor` and `_pressedTextColor`.

You should also notice that the push button component has three events, which are triggered in response to a change in state, invoked by the user: `onFocusIn`, `onFocusOut`, and `onSelect`. In Listing 4-1 you should be able to see that each event has a handler.

When you run the example, you'll also see that each button state is represented by a different background color and text labels, so when you select or press each button, the text changes according to the state.

Figure 4-3 shows a screenshot of Device Central with the push button example running.



FIGURE 4-3

Next, we'll take a short look at the soft keys UI component.

www.it-ebooks.info

USING SOFT KEYS

Soft keys are a common feature on the majority of mobile devices that do not have a dedicated touch screen interface.

The Sony Ericsson soft key UI component contains three buttons and maps the key presses of the mobile device to the left soft key, right soft key and the enter key (or select key). It is made up of `MovieClip` assets and labels, which can be styled according to the state that they are in.

Figure 4-4 shows the Soft key component example running in Device Central.



FIGURE 4-4

Listing 4-2 demonstrates how you can utilize the SoftKeys component on the timeline of a .fla file. The example snippet requires that you have the soft key component in your library, with the linkage name "SoftKeys". The example demonstrates how you can disable the left and right soft keys by first setting the text on these to a blank string "", and by setting their enabled states `_isLSKEnabled` and `_isRSKEnabled` to `false`. The only enabled soft key shown in the example is the middle key, which has the text label `_MSK` set to "Select". This key also has its text color value `_MSKTextColor` set to white `0xFFFFFF`.

www.it-ebooks.info



LISTING 4-2: Using the SofyKeys UI component

```
this.attachMovie("SoftKeys", "softKeys", this.getNextHighestDepth());

softKeys._isLSKEnabled = false;
softKeys._LSK = "";
softKeys._isRSKEnabled = false;
softKeys._RSK = "";
softKeys._MSK = "Select";
softKeys._MSKTextColor = 0xFFFFFF;

softKeys.onSoftKeyDown = onSoftKeyDownHandler;

function onSoftKeyDownHandler(keySelected) {
      trace("The selected Soft Key is " + keySelected);
}
```

The soft key component essentially acts as a multiple button component that has one key press event handler, called `onSoftKeyDown`. In this example, the defined `onSoftKeyDownHandler()` method returns the `keySelected` parameter when either of the keys is pressed. This value is set to either "LSK", "MSK", or "RSK". For this example only "MSK" will be returned, because the other buttons are disabled.

USING STATUS BAR

Using the status bar UI component allows you to simply add the status of the network connection, a battery life indicator, and the time to your applications.

Listing 4-3 demonstrates attaching the status bar component in the first frame of the timeline of a .fla file.



LISTING 4-3: Using the StatusBar UI component

```
this.attachMovie("StatusBar", "statusBar", this.getNextHighestDepth());

statusBar._textColor = 0xFFF000;
statusBar._isBackgroundVisible = true;
statusBar._timeFormat = "am-pm";
statusBar._backgroundColor = 0xFFFFCC;
```

There are limitations on styling parts of the component; you can set the text color for the time `_textColor` and the background color `_backgroundColor`. However, you're not able to change the color properties of the individual bars for network signal or battery indicator.

The component calls the `setInterval()` method to update the time, network signal, and battery capacity, making `fscommand2()` command calls to retrieve the signal level, power source, battery level, and the time. There may be a performance tradeoff when using this component, so use it wisely.

www.it-ebooks.info

The background for the status bar component can be hidden by setting the `_isBackgroundVisible` property to `false`. The time format property `_timeFormat` can be changed so that it is displayed in either "24hr" or "am-pm" format.

Figure 4-5 shows the status bar component in Device Central.



FIGURE 4-5

When testing this component in Device Central, be sure to experiment with the Device Status and Network Status Panels in the emulator. Here, you can change the battery and signal level, and then see the output of the component in the device emulator.

USING TITLE

The Title component defines the header of an application underneath the status area. It is used mainly to display the section or category heading of an application where users may need help in identifying which part of the application they are interacting with. There are examples of usages of this component throughout this chapter.

USING CHECK BOXES

The check box is effectively a data control button that can have one of two values at any particular time, `true` or `false`. Check boxes are commonly used as part of applications where the end user is able to select multiple options in one view. Use check boxes to capture data from the user.

www.it-ebooks.info

The check box component is perfect for presenting a custom setting in an application where you want the user to be able to select from a number of possible choices that are independent. A pizza-ordering application where users are able to choose a list of toppings is a common example of check box usage.

Another example usage of check boxes is a mobile MP3 player application, where you could present the user with optional settings to determine how media is played back:

- **Shuffle** — An option that lets the user play the songs in a random order.
- **Continuous play** — An option that lets the user play the next track at the end of each song.

As you can imagine these options are independent of each other, and each setting would have its own purpose.

Listing 4-4 demonstrates the usage scenario for the Check Box component; the code should be added to the first frame of the timeline of a .fla file.



LISTING 4-4: Using the CheckBox UI component

```
fscommand2("FullScreen", true);

this.attachMovie("Title", "statusTitle", this.getNextHighestDepth());

statusTitle._text = "Mp3 Player Settings";
statusTitle._textColor = 0xFFFFFF;
statusTitle._backgroundColor = 0x000000;

var blueStyle:Array = [0x074A81,
                       0x074A81,
                       0x5599BB,
                       0x5FA1C2,
                       0x074A81,
                       0xB3D2F7,
                       0x333333];

var checkBoxes:Array = new Array();

addCheckBox( "continousPlay", "Continous Play", blueStyle, 5, 65);
addCheckBox( "shuffle", "Shuffle", blueStyle, 5, 105);

function addCheckBox(cbName, cbText, cbStyle, x, y) {

      this.attachMovie("CheckBox", cbName, this.getNextHighestDepth());

      this[cbName]._x = x;
      this[cbName]._y = y;
      this[cbName]._labelText = cbText;
      this[cbName]._disabledHighlightTextColor = cbStyle[0];
      this[cbName]._disabledTextColor = cbStyle[1];
      this[cbName]._highlightColor = cbStyle[2];
```

continues

www.it-ebooks.info

LISTING 4-4 *(continued)*

```
        this[cbName]._highlightSelectedColor = cbStyle[3];
        this[cbName]._highlightTextColor = cbStyle[4];
        this[cbName]._selectedColor = cbStyle[5];
        this[cbName]._textColor = cbStyle[6];

        this[cbName].onFocusIn = onFocusInHandler;
        this[cbName].onFocusOut = onFocusOutHandler;
        this[cbName].onSelect = onSelectHandler;

        checkBoxes.push( this[cbName] );
}

function onSelectHandler(target:String, value:Boolean, cbText:String){
        statusTitle._text = cbText + " (" + value + ")";
        statusTitle._textColor = 0xFFFFFF;
}

function onFocusInHandler(target:String, value:Boolean, cbText:String){

}

function onFocusOutHandler(target:String, value:Boolean, cbText:String){
        statusTitle._text = "Mp3 Player Settings";
}
```

In this example there are two check boxes created via the `addCheckBox()` method. The method takes five parameters: an instance name, `cbName`; a label, `cbText`; a color theme, `cbStyle`, which contains an array of colors; and lastly the x and y positions for placement of the check box. The instance name, label text, and positions we've covered in the button section.

Each check box is added to the stage through the `attachMovie()` method, which uses the Linkage Identifier "CheckBox". It's presumed that you have the component in the library.




```
this.attachMovie("CheckBox", cbName, this.getNextHighestDepth());
```

code snippet chapter4_CheckBox.fla

The example also uses another component of the Sony Ericsson UI Component Library, the Title component, to display the result of the check box selections. Like the push button, the check box has a number of style properties that can be set to differentiate between its different states to set the color of the button.

In Listing 4-4 take a look at the methods that handle the check box events, `onFocusInHandler()`, `onFocusOutHandler()`, and `onSelectHandler()`. You'll notice each handler function for the event has three paramaters, a reference to the instance that is calling the method, `target`. Next is the value of the check box `cbValue`, returned as either `true` or `false`, and the third parameter is the text value of the check box `cbText`. The `onSelectHandler()` method defined in Listing 4-4 also demonstrates the usage of the check box and, in particular, how to retrieve its value when selected and setting it in the title UI component.

www.it-ebooks.info

USING RADIO BUTTONS

The radio button is another data control that can be used for retrieving input from a list of options. Unlike the check box, a radio button must be grouped with other radio buttons, which are collectively defined by their group name or related category. Typically, each radio button should have its own unique value and label to distinguish it from the other radio buttons in the group and also to present the user with the choice of selecting one option out of a possible two or more.

Take the selection of paint color as an example. You can present users with an array of colors to choose from, but you only want to give them the option of selecting one color.

Listing 4-5 demonstrates how to use the Radio Button component.



LISTING 4-5: Using the RadioButton UI component

```
this.attachMovie("Title", "statusTitle", this.getNextHighestDepth());

statusTitle._text = "Blank Canvas";

var value1:Array = [0xC0DEED,
                    0x074A81];

var style1:Array = [0x074A81,
                    0x5599BB,
                    0x5FA1C2,
                    0xB3D2F7,
                    0x333333];

var value2:Array = [0xFF9D9D,
                    0xE80000];

var style2:Array = [0xFFA8A8,
                    0xE80000,
                    0xE80000,
                    0xFF9D9D,
                    0x333333];

var value3:Array = [0xFFFF99,
                    0xFFDA09];

var style3:Array = [0xFFDA09,
                    0xFFDA09,
                    0xFFDA09,
                    0xFFDA09,
                    0xFFDA09,
                    0xFFFF99];

var value4:Array = [0x8DDC48,
                   0x66Cc04];
```

continues

www.it-ebooks.info

LISTING 4-4 *(continued)*

```
var style4:Array = [0x8DDC48,
                    0x66Cc04,
                    0x66Cc04,
                    0x66Cc04,
                    0x66Cc04];

function addRadioBtn(rbGroup:String, rbName:String, rbText:String, rbValue:Array,
rbStyle:Array, x:Number, y:Number) {

      this.attachMovie("RadioButton", rbName, this.getNextHighestDepth());

      this[rbName]._x = x;
      this[rbName]._y = y;
      this[rbName]._labelText = rbText;

      this[rbName]._group = rbGroup;
      this[rbName]._value = rbValue;

      this[rbName]._backgroundColor = 0xFFFFFF;
      this[rbName]._disabledHighlightTextColor = rbStyle[0];
      this[rbName]._disabledTextColor = rbStyle[0];
      this[rbName]._highlightColor = rbStyle[1];
      this[rbName]._highlightSelectedColor = rbStyle[2];
      this[rbName]._highlightTextColor = rbStyle[0];
      this[rbName]._selectedColor = rbStyle[3];
      this[rbName]._textColor = 0x000000;

      this[rbName].onSelect = onSelectHandler;

}

function onSelectHandler(target:MovieClip, rbGroup:String, rbLabel:String, rbValue:
String){
      var myColor = new Color("wallpaper");
      myColor.setRGB(Number(rbValue[0]));

      statusTitle._backgroundColor = Number(rbValue[1]);
      statusTitle._text = "Canvas is " + rbLabel.toLowerCase();
      statusTitle._textColor = 0xFFFFFF;
}


addRadioBtn("rbColors", "rbOne", "Blue", value1, style1, 5, 65);
addRadioBtn("rbColors", "rbTwo", "Red", value2, style2, 5, 105);
addRadioBtn("rbColors", "rbThree", "Yellow", value3, style3, 5, 145);
addRadioBtn("rbColors", "rbFour", "Green", value4, style4, 5, 185);
```

In Listing 4-5 you can see that the radio buttons are added to the stage via the `addRadioBtn()` method, which takes seven parameters. Here, it's important to note the radio button group name,

www.it-ebooks.info

rbGroup, which is the same for each radio button when it is created, rbColors, and the value assigned to the radio button, rbValue, which is unique. Both of these parameters are supplied to the addRadioBtn() method. The other parameters, such as rbName, rbText, rbStyle, x, and y, are similar to parameters that you've covered in the previous sections.

In this example, each radio button is added to the stage using attachMovie() from the library.




```
this.attachMovie("RadioButton", rbName, this.getNextHighestDepth());
```

code snippet chapter4_RadioButton.fla

Each radio button is grouped together using the _group property, and the value is assigned through the _value:




```
this[rbName]._group = rbGroup;
this[rbName]._value = rbValue;
```

code snippet chapter4_RadioButton.fla

The onSelectHandler() method for the radio button, which is invoked when the user selects each radio button, has four parameters in total, including the group name rbGroup and the value rbValue, the label defined as rbLabel, and a reference to the target MovieClip for the call, named target.




```
function onSelectHandler(target:MovieClip, rbGroup:String, rbLabel:String, rbValue:
Array){
      var myColor = new Color("wallpaper");
      myColor.setRGB(Number(rbValue[0]));

      statusTitle._backgroundColor = Number(rbValue[1]);
      statusTitle._text = "Canvas is " + rbLabel.toLowerCase();
      statusTitle._textColor = 0xFFFFFF;
}
```

code snippet chapter4_RadioButton.fla

Here, the rbValue represents an array of numbers. This is used to set the color of the movieClip instance "wallpaper", which is present on the stage of the .fla file.

As with the check box, the radio button has style properties; there are an additional four state-related styles for the radio button, making nine in total:

- **Selected** — When a radio button is selected.
- **Selected and highlighted** — When a radio button is selected and highlighted.
- **Selected and disabled** — When a radio button is selected and disabled.
- **Selected, disabled, and highlighted** — When a radio button is selected, disabled, and highlighted.

www.it-ebooks.info

When the user selects a radio button, its state changes to *selected*. Then when the user selects another radio button in the group, the previously selected button is deselected. Selecting one radio button should deselect all the other selected radio buttons in a group.

In the next section we'll take a look at the list component.

USING LISTS

The List UI component includes the functionality to structure and present data in rows and enables a user to select and navigate vertically through items in the list by using the up and down keys. Application menus are a perfect example of where the list component can be used.

The SE UI Components has numerous variations of the list component:

- Single row lists (with and without icons)
- Dual row lists (with and without icons)
- Lists with check boxes
- List with radio buttons

Each variation of the list component has two events that are dispatched:

- `onFocusChange` — Triggered when the user navigates to an item in the list.
- `onSelect` — Triggered when the user selects an item in the list.

Creating a Single Row List

Here, you can follow the steps to create an example of using lists with the single row list component. In the example, you'll define a category of film genres for the list, and when each item is selected, it will display the films under that genre.

1. First ensure that the application will run in full-screen mode when it is launched and that the stage is prevented from scaling (see Listing 4-6).



LISTING 4-6: Setting the FullScreen and scaleMode properties for the .fla file

```
fscommand2("FullScreen", true);
Stage.scaleMode = "noScale";
```

2. Next, create an instance of the title `statusTitle` setting the text to "Film Genres" (see Listing 4-7).



LISTING 4-7: Attaching the Title UI component

```
this.attachMovie("Title", "statusTitle", this.getNextHighestDepth());
statusTitle._text = "Film Genres";
```

3. Next, create a list of genres in the `genreList` array (see Listing 4-8).

www.it-ebooks.info



LISTING 4-8: Creating a list of genres for the Single Row List UI component

```
var genreList:Array = ["Action",
                       "Adventure",
                       "Animated",
                       "Biography",
                       "Childrens",
                       "Comedy",
                       "Crime",
                       "Disaster",
                       "Documentary",
                       "Drama",
                       "Fantasy",
                       "Horror",
                       "Musical",
                       "Mystery",
                       "Romance",
                       "Sci-Fi",
                       "Short",
                       "Sport",
                       "Thriller",
                       "War",
                       "Western" ];
```

4. Then, create an instance of the single row list called `srList`, and ensure that the position of the list is 58 pixels along y. This should place it directly beneath the title component (see Listing 4-9).



LISTING 4-9: Attaching the Single Row List UI component to the stage

```
this.attachMovie("ListSingleRow", "srList", this.getNextHighestDepth());
srList._x = 0;
srList._y = 58;
```

5. Next assign handlers for the `onFocusChange` and `onSelect` events (see Listing 4-10).



LISTING 4-10: Assigning an event handler to the onFocusChange and onSelect events

```
srList.onFocusChange = onFocusChangeHandler;
srList.onSelect = onSelectHandler;
```

6. Next assign `genreList` as the data provider for the list (see Listing 4-11).



LISTING 4-11: Assigning genre list to the _text property of the Single Row List UI Component

```
srList._text = genreList;
```

www.it-ebooks.info

7. Next, create an array of movies for each genre listed and save this to another data array, called `filmList` (see Listing 4-12).



LISTING 4-12: Using the RadioButton UI component

```
var action:Array = ["Heat"];
var adventure:Array = ["Raiders of the Lost Ark"];
var animated:Array = ["Toy Story"];
var biography:Array = ["Ali"];
var childrens:Array = ["Willy Wonka and the Chocolate Factory"];
var comedy:Array = ["Coming to America"];
var crime:Array = ["The Departed", "The Godfather"];
var drama:Array = ["The Color Purple"];
var fantasy:Array = ["Lord of the Rings: Fellowship of the Rings"];
var horror:Array = ["The Shining"];
var musical:Array = ["South Park: The Movie"];
var mystery:Array = ["Mulholland Drive"];
var romance:Array = ["When Harry Met Sally"];
var sciFi:Array = ["Star Wars V: The Empire Strikes Back"];
var short:Array = ["The Empire Strikes Back"];
var sport:Array = ["Jerry Maguire"];
var thriller:Array = ["Silence of the Lambs"];
var western:Array = ["Back to the Future: Part 3"];

var filmList:Array = [action,
                      adventure,
                      animated,
                      biography,
                      childrens,
                      comedy,
                      crime,
                      drama,
                      fantasy,
                      horror,
                      musical,
                      mystery,
                      romance,
                      sciFi,
                      short,
                      sport,
                      thriller,
                      western];
```

8. Define the `onFocusChange` event handler, to save the index of the current item in the list. You'll also need to declare a variable `selectedIndex` to save the value (see Listing 4-13).



LISTING 4-13: The onFocusChangedHandler method

```
function onFocusChangedHandler(){
      selectedIndex = srList.getFocusIndex();
}
```

www.it-ebooks.info

9. Define the `onSelectHandler()` method to populate list with the selected genre (see Listing 4-14).



LISTING 4-14: The onSelectHandler method

```
function onSelectHandler(){
      srList.removeAll();
      srList._text = filmList[selectedIndex];
      srList.onSelect = null;
      srList.onFocusChange = null;
}
```

If you run the application, you will be able select films from the different genres specified in the `genreList` array.

You will notice here that the `_text` value for the single row list component takes an array of strings as the data provider. Each value in the array represents a different row in the list.

If the `_text` value isn't specified, the list reverts to a default value `_emptyListText` and displays this as a single item at the top of the list. This is especially useful in a situation where you are waiting for `_text` value to be set, after you have added the list to the stage.

As with the other components, each type of list has a number of styles associated with the state, including "enabled," "highlighted," and "disabled" states. The disabled state can also be set by specifying that `_isEnabled` is `false`. As mentioned, the list component just has the two events, `onFocusChange` and `onSelect`. The `onFocusChange` event is invoked when the user moves from one row in the list to another.

When the focus changes in the list, it automatically updates the `_highlightPosition` property of the list. This is demonstrated by the `onFocusChangeHandler()` method, which updates the title component when you move up and down the list. `_highlightPosition` can also be set to position the initial focus of the list. This would be useful in a situation where you had a series of menus and wanted to present the user with the last selected menu item in the list.

The `onFocusChangeHandler()` event handler demonstrates the usage for two of the methods available for each type of list component: `getFocusIndex()` and `getListLength()`. Both the values returned by these methods are used in the title component. The focus index value returned by `getFocusIndex()` is essentially the `_highlightPosition` minus 1; it is the zero-based index value.

The list component has two additional visual features worth mentioning:

- **Scroll bar** — A track and slider that indicates how much content extends outside the view on the list.
- **Separator** — A horizontal line separating each row in the list.

The scroll bar forms part of many of the UI components in the Sony Ericsson library, and consists of the main track, the track outline, and the slider. Each style can be set through setting its respective color property. If you want, you can also remove the scroll from view by setting the `_scrollbarVisibility`.

www.it-ebooks.info

The separator consists of a single line color and a shadow color; they can be specified by setting `_separatorColor` and `_separatorShadowColor`, respectively.

Returning to the film genre example, the user would be able to select a film genre from the list and then navigate to the desired film.

Dual Row Lists and Lists with Icons

The dual row list component is another variant of the list component that can have two rows of data per list item instead of just the one.

The main difference between creating the single row list and dual row list is that you need to set two array data providers to the component instead of just the one. The property `_text1` represents the first row in the list, while `_text2` represents the second row.

Listing 4-15 provides an example for creating and providing data to a dual row list component named `drList`, with an icon defined for each row.



LISTING 4-15: Using the ListTwoRowWithIcon UI component

```
fscommand2("FullScreen", true);
Stage.scaleMode = "noScale";

var socialNetworks:Array = ["Twitter",
                            "Facebook",
                            "MySpace"];

var iconsList:Array = ["icons/Twitter_48x48.png",
                       "icons/Facebook_48x48.png",
                       "icons/MySpace_48x48.png"];

this.attachMovie("Title", "statusTitle", this.getNextHighestDepth());
statusTitle._text = "Social Networks";

this.attachMovie("ListTwoRowWithIcon", "drList", this.getNextHighestDepth());
drList._x = 0;
drList._y = 58;

drList._text1 = socialNetworks;
drList._text2 = iconsList;

drList._icon = iconsList;
drList._iconSize = "large";
```

You can set only one of the rows to be scrollable; you do so by setting the `_scrollingTextRow` parameter to either 1 or 2. This is enabled only if you have the `_isScrollingText` property set to `true`.

The main feature of the dual list component is the ability to include local paths to icons, which you will see by running the example in Listing 4-15. There is also a single list version of the list

www.it-ebooks.info

component available. For both types of lists, there are three properties associated with adding icons or small graphics. To define these icons, you need to set the `_icon` property for the list as shown in Listing 4-15. This should be an array that contains all the file paths to the icons you want to display in each row. You can also choose which row you want to place the icon onto, by defining the `_iconPlacement` value as either `row1` or `row2`. Lastly, `_iconSize` lets you define whether the icons should be displayed as `large` or `small`.

Lists with Check Boxes

Having already covered the check box component, you may have noticed that you had to manually place each check box in position on the stage. One will admit this is rather awkward and fiddly. The check box list component takes care of this for you and makes it a lot easier for you to add check boxes using the `addItems()` method or remove one from the list through calling `remove()`.

Here's a code snippet for creating a check box list component, named simply `mp3Settings`; again, the component needs to be in the library:




```
this.attachMovie("ListCheckBox", "mp3Settings", this.getNextHighestDepth());
mp3Settings._x = 0;
mp3Settings._y = 58;
mp3Settings.addItems( ["Continous Play", "Shuffle", "Loop"] );
```

code snippet chapter4_CheckBoxList.fla

Lists with Radio Buttons

Like the check box, the radio button also has a list version. This can also be added in the same way (see Listing 4-16).



LISTING 4-16: Using the ListRadioButton UI component

```
this.attachMovie("ListRadioButton", "motownQuiz", this.getNextHighestDepth());
motownQuiz._x = 0;
motownQuiz._y = 58;

var dataProvier:Array = ["Marvin Gaye",
                         "Stevie Wonder",
                         "Jackson 5"];

var enabledStates:Array = [true,
                           true,
                           true];

motownQuiz.addItems(dataProvier, enabledStates);
```

The example in Listing 4-16 allows you to see that the `dataProvider` supplies the data for the radio button list via the `addItems()` method. You should also note that the second parameter for

www.it-ebooks.info

the method is an array of Boolean values that set the state of the radio button to either `enabled` or `disabled`. It's worth mentioning that the second parameter defines whether each radio button check box is ticked or not. The main difference between using the radio button list and the individual component is that you don't need to define the group for each radio button instance. The other difference is that you cannot retrieve the value for the radio button list.

USING MODAL DIALOGUES

The modal dialogue is a view layer component that provides the user with information. The hierarchical position of a modal dialogue is commonly found at the uppermost layers of an application, and in addition to providing information, dual dialogues serve the purpose of preventing users from continuing their interaction with the rest of the application while the dialogue is shown.

Dialogues are particularly useful in situations where you need the user to respond to a message or piece of information before allowing them to continue with the application.

The modal dialogue component was designed to cater to various situations in response to user actions in an application, including:

- **Confirmations** — Confirming task completion.
- **Errors** — Responding to an error caught by the application.
- **Information** — Displaying general information.
- **Questions** — Asking a question.
- **Warnings** — Providing a warning for a potential risk.

Generally, the dialogue component should contain information for the end user to acknowledge. Listing 4-17 demonstrates how to use the dialogue component and has data defined for each of the situations mentioned. You need to ensure that you have a Dialogue component in your library and use the following ActionScript to attach the pre-compiled movie to the stage:



LISTING 4-17: Using the Dialogue UI component

```
this.attachMovie("Title", "statusTitle", this.getNextHighestDepth());
statusTitle._text = "Modal dialogues";

var cParams:Array = ["Confirmation", "Pairing with JGANDERSON complete."];
var eParams:Array = ["Error", "Unable to find the device specified. Try again?"];
var iParams:Array = ["Information", "Information is now available on touch screen
                      devices, see
                                help section for more  details"];
var qParams:Array = ["Question", "Would you like to install new updates now?"];
var wParams:Array = ["Warning", "Battery is running low, please charge device
                      now!"];

var dialogueParams:Array = [cParams, eParams, iParams, qParams, wParams];
var isDialogueVisible = false;
```

www.it-ebooks.info

```
this.attachMovie("ListSingleRow","dialoguesList", this.getNextHighestDepth());
dialoguesList._y = 58;
dialoguesList._text= ["Confirmation", "Error", "Information", "Question", "Warning"];

dialoguesList.onFocusChange = onFocusChangeHandler;
dialoguesList.onSelect = onSelectHandler;

this.attachMovie("dialogue", "dialogue", this.getNextHighestDepth());
dialogue.onOpen = onOpenHandler;
dialogue.onClose = onCloseHandler;

function onOpenHandler (){
      isDialogueVisible = true;
}

function onCloseHandler (){
      isDialogueVisible = false;
}

this.attachMovie("SoftKeys", "softKeys",  this.getNextHighestDepth());

softKeys._isLSKEnabled = false;
softKeys._LSK = "";
softKeys._isRSKEnabled = false;
softKeys._RSK = "";
softKeys._MSK = "Select";
softKeys._MSKTextColor = 0xffffff
softKeys.onSoftKeyDown = onSoftKeyDownHandler;

function onSoftKeyDownHandler(){

      var selectedParams = dialogueParams[dialoguesList.getFocusIndex()];

      dialogue._titleText = selectedParams[0];
      dialogue._icon = selectedParams[0];
      dialogue._bodyText = selectedParams[1];

      if(isdialogueVisible){
            dialogue.hide();
      } else {
            dialogue.show();
      }
}
```

The example uses four UI components: a title, soft keys, a single row list, and the dialogue. The list is used to hold each type of dialogue, and when you press select, the soft key handler activates the show method of the dialogue. For the dialogue component, you specify the main body text, `_bodyText`, for the user to read and an accompanying title, `_titleText`. The types of dialogue that we have mentioned are defined by the `_icon` property; this sets the default graphic for the icon, and here's where we also distinguish between the different types of dialogue by setting the value to "Confirmation," "Error," "Warning," "Information," or "Question." In the example, the `dialogueParams` value, which is an array of arrays holding each value, is retrieved

www.it-ebooks.info

through the focus index of the list. The `_titleText`, `_icon`, and `_bodyText` are all set by the `selectedParams` value when the middle soft key has been selected.

By default, the dialogue is hidden; to activate it you call the `show` method. The example here uses the `onClose` and `onOpen` event handler methods to define the state of the dialogue `isDialogueVisible`, to decide whether to hide it once it's open. You call the `hide` method of dialogue to remove it from screen. The `onClose` method is triggered when the dialogue box closes, while `onOpen` is triggered when the dialogue box opens.

You can define the alignment of the dialogue box by setting the `_placement` property, which can be placed either at the "center" or "bottom" of the application relative to the stage's y axis.

It's not covered here, but you can also include custom images in the dialogue; by setting the `_inlinePicture` property, you can place an image between the title and body text, or you can define custom icons by setting `_customIcon` and `_badge` settings. To remove any images you've loaded into the dialogue, you need to call `unloadImages()` method.

Lastly, you'll notice the dialogue has a transparent grey background color. This can also be styled by setting the `_dimColor` property.

USING VISUAL INDICATORS

Indicators provide a visual reference to something that is currently taking place during the activity of a mobile application. There are two Sony Ericsson UI components readily available that you can use in your applications, the Progress Indicator and the Wait Indicator.

The Progress Indicator

When you want to present the user with a precise representation of progression, use the progress indicator. This component should be used when you know the *duration* of the process taking place; for example, this could be the total number of bytes for a file download or the total length of a media track. These are definitive processes for which the progress can be measured.



You can only use progress indicators in situations where you know the minimum and maximum values of progression.

You have the option of setting the indicators as either a *bar* or *circle*, and the component has the following events that you can create event handlers for:

- `onComplete` — When the progress is complete.
- `onProgress` — When the progress value changes, you can notify the user.

The component has the following methods:

- `getProgress()` — Returns the value progress.
- `setProgress()` — Sets the progress and displays the new value.

www.it-ebooks.info

- `getWidth()` — Returns the width of the component.
- `setWidth()` — Sets the width of the component.

Listing 4-18 shows an example of how the progress indicator simulates a named `progressBar` in action using a timer `setInterval()` to increment the progress indicators through the `setProgress()` method.



LISTING 4-18: Using the ProgressIndicator UI component

```
this.attachMovie("Title", "statusTitle", this.getNextHighestDepth());

statusTitle._text = "Progress Indicator";
statusTitle._textColor = 0xFFFFFF;
statusTitle._backgroundColor = 0x387796;

this.attachMovie("ProgressIndicator", "progressBar", this.getNextHighestDepth());

progressBar._x = 30;
progressBar._y = 100;
progressBar._width = 200;
progressBar._backgroundColor = 0xffffff;
progressBar._effectColor = 0x074A81;
progressBar._fillColor = 0x5FA1C2;
progressBar._hasEffect = true;
progressBar._isPercentageText = true;
progressBar._maximum = 100;
progressBar._minimum = 0;
progressBar._shape = "Bar";

progressBar.onComplete = onCompleteHandler;

function onCompleteHandler(){
      clearInterval(progressBarntervalID);
}

var pIntervalID:Number;
pIntervalID = setInterval(updateProgress, 100);

var pValue:Number = 0;

function updateProgress(){
      progressBar.setProgress(pValue++);
}
```

When you run this example, you'll see the progress indicator incremented from 0% to 100%. This is achieved by updating the `_value` property and calling the `setProgress()` method. The `_maximum` property of the progress indicator is set to 100, and `_minimum` is set to 0.

You have the option of displaying the percentage text for the progress indicator through the `_percentageText` property.

www.it-ebooks.info

The value displayed in the percentage text is determined by a calculation of the `_value`, `_minimum`, and `_maximum` properties of the progress indicator. If you don't want to display the percentage text, you set the `_isPercentageText` value to `false`.

With the progress indicator you can additionally set an effect color property called `_effectColor`. This is only displayed when the `_shape` is set to "bar." You can turn off the effect by setting `_hasEffect` to `false`.

The Wait Indicator

The wait indicator similarly presents the user with a graphical representation to indicate that a process is taking place in the application, but over an indeterminable amount of time.

You use the wait indicator instead of the progress indicator when a process in your application is taking an *unknown* amount of time. You cannot set the progress of the wait indicator, and it doesn't have a `_minimum` or `_maximum` properties.

Listing 4-19 details a snippet of code that demonstrates how to attach the `WaitIndicator` from the library to the stage and style it as a circle.



LISTING 4-19: Using the WaitIndicator UI component

```
this.attachMovie("WaitIndicator", "waitCircle", this.getNextHighestDepth());

waitCircle._x = 15;
waitCircle._y = 100;
waitCircle._backgroundColor = 0x074A81;
waitCircle._fillColor = 0x5FA1C2;
waitCircle._shape = "Circle";
```

Both the `WaitIndicator` and `ProgressIndicator` have the `_shape` property, which allows you to define whether the component appears as a "bar" or "circle." There are no states for either type of indicator, but you can set the fill property, called `_fillColor`, and the background property, called `_backgroundColor`.

With both components you can also set and retrieve the width of the components through the `setWidth()` and `getWidth()` methods, which is especially handy when you want to limit the space the component occupies.

USING SLIDERS

The slider UI component is a visual control that represents a specific value over a data range. Like the progress indicator, the slider component has a range of values. The slider consists of a slider track and slider thumb, where the length of the slider track effectively visually represents the data range.

The value of the slider changes in accordance with the position of the slider thumb in relation to the length of the slider track. By pressing the left and right keys on the mobile device, the user can move the slider thumb and change the value it represents.

www.it-ebooks.info

An example of where a slider would be a good component for controls is in a media application to allow the user to control the volume or to change the brightness of the screen. These examples highlight how powerful the slider control is. Users don't have to think about what value they want to key in to set the brightness or volume. They just have to know how to move the slider.

In the Listing 4-20 you'll see an example of the slider component in action. Here, movement of the slider left and right alters the size of a square in the middle of the stage in the application.



LISTING 4-20: Using the Slider UI component

```
this.attachMovie("Slider", "hSlider", this.getNextHighestDepth());

hSlider._x = Math.round(Stage.width/2 – hSlider._width/2);
hSlider._y = Stage.height - 80;
hSlider._steps = 20;
hSlider._initialPosition = 10;
hSlider._isFilled = true;
hSlider._isBackgroundVisible = true;
hSlider._backgroundColor = 0xFFFFFF;
hSlider._sliderColor = 0x5FA1C2;
hSlider._sliderHighlightColor = 0x5FA1C2;
hSlider._trackColor = 0x074A81;

hSlider.onIncrease = onSliderIncrease;
hSlider.onDecrease = onSliderDecrease;

function onSliderIncrease(sender:Object, position:Number){
      resizeSquare(position);
}

function onSliderDecrease(sender:Object, position:Number){
      resizeSquare(position);
}

function resizeSquare(size:Number) {
      square._x = Math.round(square._x);
      square._y = Math.round(square._y);
      square._width = Math.round(size*5);
      square._height = Math.round(size*5);
}
```

The data range for the slider is set through the `_steps` property; here, this is set to 20. This means that the user is able to increment the slider 20 times, or from 0 to 20 from left to right horizontally. The `_intialPosition`, which is set at 10, means that the starting position for the slider thumb will be halfway along the slider track. This is a useful property to know as it means that, if you were to include this component as part of the media application, you could set the volume at a reasonable level as supposed to 0 where you would expect it to be muted.

There are four styles that you can alter for the component including background color `_backgroundColor`; slider enabled and slider color, `_sliderColor`; slider highlight color, `_sliderHighlightColor`; and also the track color, `_trackColor`. You can also hide the fill by setting `_isFilled` to `false`, or hide the background by setting `_isBackgroundVisible` to `false`.

www.it-ebooks.info

By default, the slider has plus and minus icons at the opposing ends of the component. These icons can be set or removed by specifying paths to `_minIconValue`, represented by the minus icon by default, and by specifying the `_maxIconValue`, represented by the plus icon by default.

The slider has two events, `onIncrease` and `onDecrease`, that are invoked when the user changes the slider thumb position. Both events handler methods return two parameters, the reference to the slider and the current step position of the slider. In Listing 4-20, notice that the `position` of the slider is used as a value for resizing the square on the stage.

The slider in this example is used horizontally, but you can alter the orientation of the slider so that it works vertically also.

USING SCROLLABLE AREAS

You briefly covered part of this topic in the lists section with the scroll bar. Scrollable areas are essentially containers that allow the user to scroll content that doesn't entirely fit on the mobile screen. There are two types of scrollbar components, one for images and the other for text.

Using the Scrollable Area Component

To include a scrollable area in your applications, you attach the "ScrollableArea" from the library and set the `_content` property. Listing 4-21 shows how you can use the Scrollable Area component to load an image called 640x480.png located in a device's file system. The component uses the `loadMovie()` call to load in content.


Available for download on Wrox.com

LISTING 4-21: Using the ScrollableArea UI component

```
this.attachMovie("ScrollableArea", "imageScroll", this.getNextHighestDepth());

imageScroll._content = "640x480.png";
imageScroll._hScroll = true;
imageScroll._vScroll = true;
imageScroll.onScroll = onScrollHandler;
imageScroll.onContentLoad = onContentLoadHandler;

function onScrollHandler(yPos:Number, maxY:Number, xPos:Number, maxX:Number){
      trace(" maxY " + maxY);
      trace(" yPos " + yPos);
      trace(" xPos " + xPos);
      trace(" maxX " + maxX);
}

function onContentLoadHandler(){
      trace("Image Loaded");
}
```

When you run the example, you'll notice that the image is oversized for the 240 x 320 stage defined for the `.fla` file, and this is intentional. This is to demonstrate the scrollable area. The image loaded

www.it-ebooks.info

into the example has a colored square at each corner. In Device Central, you press the keys on the directional pad to scroll to each edge of the image.

In Listing 4-21 you can see that the `onScrollHandler()` event handler method returns several parameters, including the current vertical and horizontal positions of the scroll, `yPos` and `xPos`, respectively. There are also parameters for the maximum scroll increments that need to be made to reach the bottom and far right of the image loaded; these are defined as `maxY` and `maxX` of the scroll as a parameter, while the `onContentLoad()` event handler returns the percentage of the content loaded.

Using the Scrollable Text Component

To include a scrollable text component in your applications, you simply attach the "ScrollableText" from the Library and set the `_text` property.

Listing 4-22 shows how you can use the Scrollable Text component and demonstrates how to scroll through text that doesn't fit in one view.



LISTING 4-22: Using the ScrollableText UI component

```
this.attachMovie("ScrollableText", "textScroll", this.getNextHighestDepth());

textScroll._hScroll = true;
textScroll._vScroll = true;
textScroll._fontSize = 14;
textScroll._frameColor = 0xCCCCCC;
textScroll._textColor = 0x000000;
textScroll._isFrame = true;
textScroll._isHTML = false;
textScroll._text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec
magna ligula, lacinia non tempor vitae, lobortis id justo. Vestibulum non est leo.
Etiam pulvinar aliquam orci, eget pretium dui aliquam sit amet. Nullam sed ipsum
malesuada nunc interdum gravida nec tristique nunc. Ut elit felis, sollicitudin
venenatis pulvinar at, mollis quis eros. Ut consequat faucibus enim, id ultrices
magna mollis eu. In tristique tempor sollicitudin. Donec sit amet tristique leo.
Pellentesque placerat, metus non condimentum tincidunt, urna erat ullamcorper
ligula, at venenatis tellus purus in dui. Nulla auctor porta ligula dictum euismod.
Aliquam accumsan dui ultricies urna laoreet adipiscing.";

textScroll.onScroll = onScrollHandler;
textScroll.onTextChange = onTextChangeHandler;

function onScrollHandler(yPos:Number, lineNo:Number, maxY:Number){
      trace(" yPos " + yPos);
      trace(" maxY " + maxY);
      trace(" lineNo " + lineNo);
}

function onTextChangeHandler(){
      trace("Text Changed");
}
```

www.it-ebooks.info

The scrollable text component has an additional `onTextChange` event fired when the `_text` property is updated. Like the scrollable area, this component has the `onScroll` event. The component has `_hscroll` and `_vscroll`, so you can choose whether or not to display the vertical and horizontal scroll bars.

The `_isHTML` and `_isFrame` Boolean properties are also defined; they determine whether the text is in HTML format and whether a frame is visible around the scrollable text area.

USING NOTIFICATION

Notification components are useful assets that allow you to present information to users without the need to restrict them from interacting with the main application. The notification is commonly used as an alert and is very similar to the dialogue component; however, this element doesn't require the user to interact with it.

Listing 4-23 demonstrates the usage for the Notification UI component, launching a notification to display the football score: "Sheffield Wednesday 8 — 0 Sheffield United".



LISTING 4-23: Using the Notification UI component

```
this.attachMovie("Notification", "notifyMe", this.getNextHighestDepth());

notifyMe._text = " Sheffield Wednesday 8 - 0 Sheffield United ";
notifyMe._placement = "center";
notifyMe._startTimeout = 1000;
notifyMe._wrapTimeout = 1000;

notifyMe.setWidth(190);
notifyMe.show();
```

The notification has the same methods as the dialogue component: `show()`, `hide()`, and `unloadImages()`. It also has two additional methods to set and retrieve the width, called `setWidth()` and `getWidth()`. In Listing 4-23 you can see that the `show()` method initiates the notification component, after setting the `_text` and `_placement` properties. The `_placement` property can be specified as `top`, `center`, or `bottom` and positions the notification alert accordingly.

By default, the notification will be displayed for five seconds, or until the user interacts with the application. Setting the `_durationType` property to "custom" will change the component to take note of the `_duration` property. The `_duration` property is a number in milliseconds that allows you to specify how long the notification is displayed. In addition to the display duration, there are specific timeout settings that define aspects of the notifications text scroll.

The following list shows the timeout-related properties that can be set on the notification:

- `_startTimeout` — Time before the text in the notification starts to scroll after it has been shown.
- `_scrollTimeout` — Time between each pixel scroll.

www.it-ebooks.info

- `_rowTimeout` — Length of time between each row scroll.
- `_wrapTimeout` — Length of time between the end of a scroll and when the scroll starts again.

There is also a `_scrollSpeed` property that defines the number of pixels moved for the text defined in `_text`. The notification's background color property, called `_backgroundColor`, and the text color property, called`_textColor`, can also be styled. You can also define an image `_icon` for the notification and set the size to "large" or "small" through the `_iconSize` property.

SUMMARY

You've now covered a whole range of Sony Ericsson UI components in this chapter. The Sony Ericsson team are constantly evolving their online library, with input from the Flash Lite and mobile developer community. For up-to-date information visit the developer website at `http://developer.sonyericsson.com/` and search for "UI Components".

While the majority of the components have been optimized for 240 x 320 screen sizes, they can be used at other screen sizes, and ultimately provide a good base for quickly developing applications and prototypes. They can be used in all Flash Lite AS2 projects from Flash Lite 2.0 onwards.

The Forum Nokia Flash Lite components, as mentioned earlier in the chapter, are also good assets to build applications with. For more information on their components, visit the Forum Nokia website's "Guide to Flash Lite Components" at `http://www.forum.nokia.com/info/sw.nokia.com/id/332b6e95-b173-4b58-8552-4639795212f9/Guide_to_Flash_Lite_Components.html`.

In this chapter you created the majority of the code for each example on the timeline. In the next chapter you'll take a look at using the PureMVC framework, to develop more structured code for your Flash mobile content, picking up from where we left off in Chapter 3.

www.it-ebooks.info

5

PureMVC ActionScript 2.0 Framework

WHAT'S IN THIS CHAPTER?

- Implementing the PureMVC framework for Flash Lite applications
- Exploring key concepts
- Creating an example PureMVC application

PureMVC is a free open source development framework that can be used to develop and structure your Flash applications. The reference implementation for the PureMVC framework was written in AS3 by project architect and the founder Cliff Hall.

For Flash Lite, you will be using the AS2 version of the framework ported by Pedr Browne.

DESIGN PATTERNS AND DEVELOPMENT FRAMEWORKS

For Flash Lite Mobile Device application developers, the options for a reliable and robust development framework are limited.

While there are numerous development frameworks for AS2, they tend to be for the desktop or browser version of the Flash Player and have not been considered or optimized for mobile device usage.

Time for a Development Framework

So why do Flash Lite mobile device applications need a development framework?

www.it-ebooks.info

Among the many aspects to development, the design of mobile applications requires you to consider what happens when the user interacts with their application on the device, when data should be retrieved, or how a view should be displayed. A development framework like PureMVC can provide us with a structured solution that enables us to implement all of the above features with ease.

The following areas give you an idea of the current obstacles to developing Flash Lite mobile device applications without a development framework.

Maintenance Headaches

If you are familiar with coding for Flash Lite players version 1.0 and 1.1, you will be more than aware that all of your ActionScript has to be placed directly on the timeline or hidden away inside or on a *MovieClip* or *Button*.

Depending on how complex the project is, one could potentially end up with a really cluttered timeline, a maintenance headache, and as a project progresses, code that becomes increasingly impossible to find and can also be difficult to debug.

In a nutshell, this is all very time consuming for developers and one of the more unwanted pleasures of Flash Lite builds. The result leaves your carefully crafted mobile application slightly tainted by disorganized source code.

Figure 5-1 shows the timeline panel for an old Flash Lite 1.1, AS1, Poker game, highlighting complex organization of code.

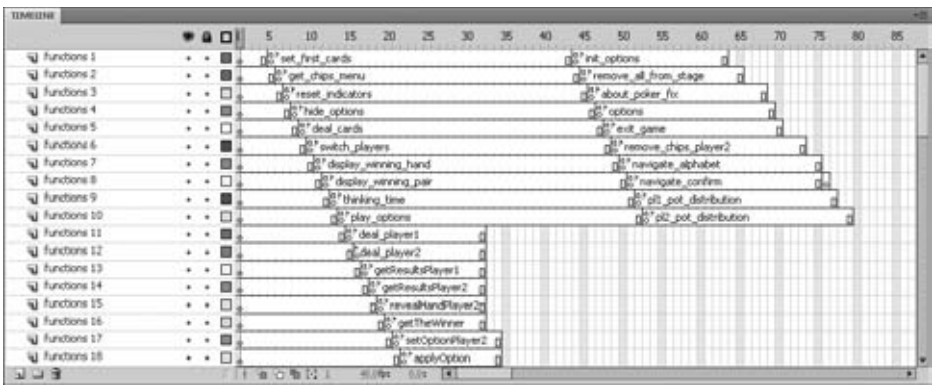


FIGURE 5-1

www.it-ebooks.info

Tight-Coupling and Specific Code Implementations

Maintenance headaches are less of a worry for developers who are accustomed to OOP for Flash Lite AS2, with the introduction of class based development supported by Flash Lite player version 2.0 onwards to the current version.

However code can still result in tightly-coupled components or objects being created. Furthermore, while components can hide a lot of their own implementations, developing complex applications can often result in particular sections of code, whether that is a component or a particular feature, being programmed to very specific implementations.

In some instances this can mean whenever you want to change one part of your code you often need to change another. A code design problem in itself, but one which also leads to a different kind of maintenance issue than that previously mentioned.

Reliability and Usability

The openness of the PureMVC project, since its conception, has allowed developers worldwide to put the robustness of the PureMVC framework to the test, and it is regarded as a reliable code base. It helps the developer to create more isolated components that we will later see can be reusable in other projects.

Its strength and popularity in the developer community has come from it being highly portable, being ported to various languages other than AS2 and AS3 including C#, ColdFusion, haXe, Java, JavaScript, Objective C, PHP, Python and Ruby. Not only does the framework cover portability for desktop software, it also has a home for mobile, browser and server environments. This is a testament to its usability in addition to reinforcing the success and wide adoption of the PureMVC project, making it a good choice for use in our Flash Lite Mobile Device Application development.

Design Patterns Used in PureMVC

PureMVC is derived from several design patterns and principles. Design patterns in their visual form display relationships and interactions between objects and are often represented as conceptual diagrams.

The framework is largely based on the Model-View-Controller (MVC) design pattern methodology where the development focus is on separating code logic into distinct parts; Model, responsible for data; View, responsible for visual objects; and Controller, responsible for the core logic, managing when the Model and View are updated. PureMVC has a key Actor known as the *Facade* which is an addition to the principles of MVC.

Figure 5-2 shows the generic PureMVC concept diagram.

www.it-ebooks.info

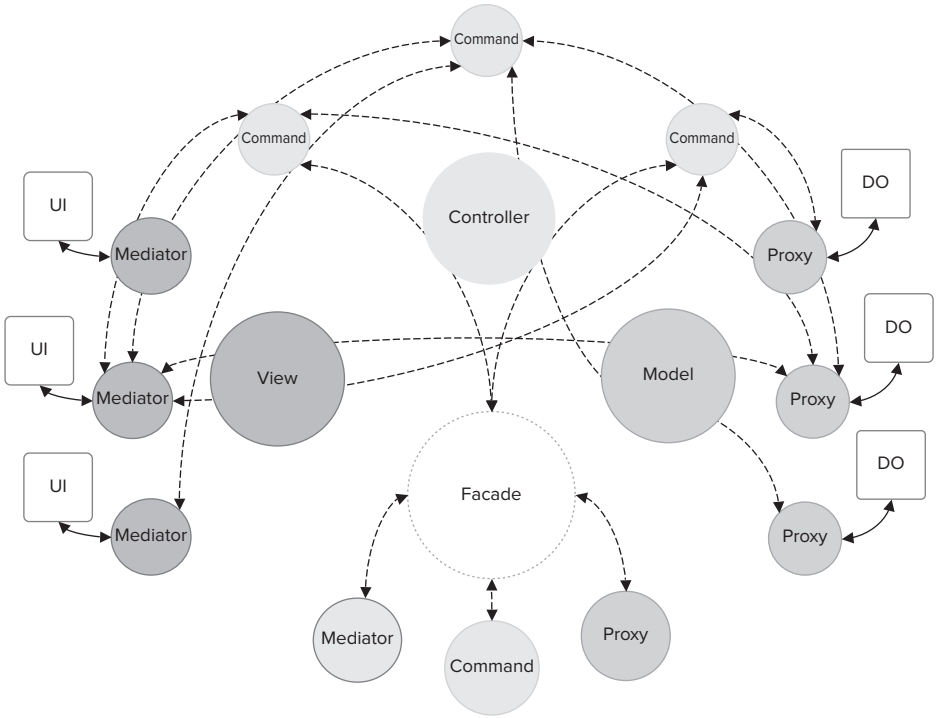


FIGURE 5-2

A key factor in strength of PureMVC lies in the proven design patterns employed by the framework. These patterns effectively help to form the rules of the framework and hence are important concepts to understand.

Table 5-1 lists the intentions that each design pattern used in the PureMVC framework aims to address.

TABLE 5-1

DESIGN PATTERN	INTENT
Singleton	Ensure a class has only one instance, and provide a global point of access to it.
Observer	Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
Factory	Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
Facade	Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

www.it-ebooks.info

DESIGN PATTERN	INTENT
Proxy	Provide a surrogate or placeholder for another object to control access to it.
Command	Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
Mediator	Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Source: Wikipedia Design Pattern Computer Science Page

PUREMVC KEY CONCEPTS EXPLAINED

There are several key concepts to grasp in getting to know the PureMVC framework, and in this section you'll go over the key concepts of PureMVC. Along the way there are some code references to follow, so you can build familiarity.



In this section there are references to the PureMVC framework along with theoretical examples which are not meant to be used in isolation. Ensure you have the PureMVC package accompanying this chapter to follow this section.

Here you'll go through each of the following key concepts in turn:

- The Facade
- Notifications and Observers
- Commands and the Controller
- Proxies and the Model
- Mediators and the View

The Facade

The `Facade` has the central role in PureMVC and is essentially responsible for ensuring your application runs like clockwork by governing the roles of the *Model*, *View* and *Controller*, managing and delegating responsibilities to each throughout the framework.

Creating ApplicationFacade

The PureMVC framework provides you with an abstract `Facade` class; in the zip file for this chapter you can locate the file here in `src/org/puremvc/as2/patterns/facade/Facade.as`.

For all your PureMVC applications you need to create a concrete implementation of the base `Facade` class. By convention this is named `ApplicationFacade`. You create `ApplicationFacade`

www.it-ebooks.info

by extending `Facade`, while implementing the interface `IFacade`. By convention you also import the interfaces and facade packages using the wild card (.*). There are three other key aspects of `ApplicationFacade` which you should familiarize yourself with, as this is where your model, view, and controller's import statements are also referenced, ready for use in the applications. Listing 5-1 shows the bare bones of the `ApplicationFacade` created from the points discussed so far.



LISTING 5-1: Creating the ApplicationFacade class

```
import com.wrox.chapter5.model.*;
import com.wrox.chapter5.view.*;
import com.wrox.chapter5.controller.*;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.facade.*;

class com.wrox.chapter5.ApplicationFacade extends Facade implements IFacade {

}
```

In Listing 5-1 notice `ApplicationFacade` doesn't have a constructor method for instantiation, this omission is deliberate. Instead of having a constructor you override the public static `getInstance()` method in `Facade` to return an instance of `ApplicationFacade`. This method is a *singleton* implementation (see Listing 5-2).



LISTING 5-2: Overriding getInstance() in ApplicationFacade

```
import com.wrox.chapter5.model.*;
import com.wrox.chapter5.view.*;
import com.wrox.chapter5.controller.*;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.facade.*;

class com.wrox.chapter5.ApplicationFacade extends Facade implements IFacade {

      public static function getInstance():ApplicationFacade {
            if (instance == null) {
                  instance = new ApplicationFacade();
            }

            return ApplicationFacade(instance);
      }
}
```

The `instance` property referenced in `getInstance()` is inherited from `Facade` and has a type of `IFacade`. When the `getInstance()` method is called through the `ApplicationFacade`, the `Facade` is instantiated and initializes the model, controller, and view, via calls to the `intializeModel()`, `initializeController()`, and `initializeView()` methods respectively, hence no constructor call

www.it-ebooks.info

in `ApplicationFacade` is required. The following snippet shows the constructor for `Facade` and the `initializeFacade()` method:




```
public function Facade() {
      if (instance != null) throw Error(SINGLETON_MSG);
      instance = this;
      initializeFacade();
}

private function initializeFacade():Void {
      initializeModel();
      initializeController();
      initializeView();
}
```

code snippet org/puremvc/as2/patterns/facade/Facade.as

Subsequently, via the concrete `Facade`, you can initialize the PureMVC framework on the timeline of your Flash document by calling the static `getInstance()` method:




```
import com.wrox.chapter5.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
```

code snippet chapter5_PureMVC.fla

initializeController()

Of the three *initialize* methods called in `Facade`, `initializeController()` is where your application initially comes together. PureMVC recommends you override the `initializeController()` function in `ApplicationFacade`. When you override this method you also need to ensure that `initializeController()` method is called in the parent class (see Listing 5-3).



LISTING 5-3: Overriding the initializeController method in ApplicationFacade

```
public static function getInstance():ApplicationFacade {
      if (instance == null) {
            instance = new ApplicationFacade();
      }

      return ApplicationFacade(instance);
}

private function initializeController():Void {
      super.initializeController();
}
```

www.it-ebooks.info

The `initializeController()` method is where you register core command calls for the application. This overlaps with our next PureMVC concept, so before completing this method we'll move onto discussing *Notifications and the Observer.*

Notifications and the Observer

Notifications are message objects that are used in the PureMVC framework to broadcast to the model, view, and controller that a significant event has taken place. Part of a message object consists of a notification name, a string whose value represents a unique event. `STARTUP` is a common notification name variable used in the concrete `Facade` as a signal to say start the application; another example is `QUIT`, a notification to indicate a request to end an application. In AS2 the notification names are written as public static variable strings, and are commonly declared in `ApplicationFacade` (see Listing 5-4).



LISTING 5-4: Declaring notification names in the ApplicationFacade

```
public static var STARTUP:String = "startUp";
public static var QUIT:String = "quit";

public static function getInstance():ApplicationFacade {
      if (instance == null) {
            instance = new ApplicationFacade();
      }

      return ApplicationFacade(instance);
}

private function initializeController():Void {
      super.initializeController();
}
```

Notification names are simply call to actions. I recommend you declare all your notification names in your `ApplicationFacade` before moving onto other parts of the application. Later you'll be adding to these notifications.

One of the roles of the `Facade` is to orchestrate the sending of notifications within the framework. In the PureMVC framework there are two important classes to note which are responsible message notifications, found in the observer package: `Notifier` and `Notification`. The `Notifier` class can be found here: `src/org/puremvc/as2/patterns/observer/Notifier.as`. The `Notification` class can be found here: `src/org/puremvc/as2/patterns/observer/Notification.as`.

The `Notification` class which implements the interface `INotification` is your message type. It has three properties: `name`, `body`, and `type`. Each property has an associated method to set and return its value. The property `name` is a string reference to the notification. The `body` is used to store a reference to an object associated with the notification and the `type` is a string representation of the `body` object `type`.

www.it-ebooks.info

The `Notifier` class is extended by the base classes of each key player within the framework; the `Mediator`, `Command`, and `Proxy` classes each inherit the methods and properties of `Notifier`.

On instantiation the `Notifier` class assigns a local reference to the `Facade` singleton:




```
facade = Facade.getInstance();
```

code snippet org/puremvc/as2/patterns/observer/Notifier.as

sendNotification()

You use the `sendNotification()` method to deliver notifications to the rest of the framework. The `sendNotification()` function in the `Notifier` class is where each aspect of the framework is able to broadcast significant events or messages to each other. But they don't do this directly. They do it through the `Facade` class:




```
facade.notifyObservers(new Notification(notificationName, body, type));
```

code snippet org/puremvc/as2/patterns/observer/Notifier.as

By doing this notifications are always coordinated though the `Facade`. When the `sendNotification()` method is called, the `Facade` calls the `notifyObservers()` method, passing it a `Notification` object, which is then added to a list of observers in an `ObserverMap` held in the initialized view.

The `notifyObservers()` method requires that `INotification`, in the form of the `Notification` object, is forwarded as a parameter:




```
notifyObservers(new Notification(notificationName, body, type));
```

code snippet src/com/wrox/chapter5/ApplicationFacade.as

The `Facade` also has an implementation of the `sendNotification()` method; this is inherited by your `ApplicationFacade` and can be utilized to start your application. Let's look at this next:




```
public function sendNotification( notificationName:String, body:Object, type:
String):Void {
      notifyObservers(new Notification(notificationName, body, type));
}
```

code snippet src/org/puremvc/as2/patterns/facade/Facade.as

Returning to `ApplicationFacade`, once you retrieve your concrete `Facade` instance you can kick start the application. It's common practice within `ApplicationFacade` to create the public method called `startup()`, which then calls the inherited `sendNotification()` method. This sends the

www.it-ebooks.info

notification name `STARTUP`, along with a reference to the root of the application, which is supplied as the `body` parameter. In the `startup()` method this is a MovieClip. Listing 5-5 shows the complete `startup()` method, demonstrating how the STARTUP message is sent.



LISTING 5-5: The startup method for ApplicationFacade

```
public function startup(app:MovieClip):Void {
      sendNotification(STARTUP, app);
}
```

On the timeline of the Flash document you call `startup()` on the `ApplicationFacade` instance:




```
import com.wrox.chapter5.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
app.startup(this);
```

code snippet chapter5_PureMVC.fla

Now that you understand how notifications are defined and sent, you need to know how they are used further and what to do with them.

You can effectively delegate a responsibility to a notification name, then when the notification is sent an *action* will be triggered. The `STARTUP` notification is commonly assigned to the command called `StartupCommand`, and as you will see in the next section this class essentially determines what happens when your application starts. This nicely leads us onto our next concept, *Commands and the Controller*.

Commands and the Controller

Commands are the directives within PureMVC that carry out a particular task. You can create any number of commands in your application to action tasks. You will learn that command names and corresponding roles are very specific and are *usually* paired with their notification name, like with the `StartupCommand` and `STARTUP`, the `QuitCommand` and `QUIT`. This makes understanding applications a lot easier when you architect them.

MacroCommand versus SimpleCommand

The PureMVC framework provides two base classes for concrete implementations of commands, `MacroCommand` and `SimpleCommand`.

You extend `SimpleCommand` when you need to create a command for carrying out a specific action, like with the `QuitCommand`. When the QUIT notification is sent the `fscommand2()` method is executed in the `QuitCommand` (see Listing 5-6).



LISTING 5-6: QuitCommand, an example of SimpleCommand

```
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.command.SimpleCommand;
```

www.it-ebooks.info

```
import org.puremvc.as2.patterns.observer.Notification;

class com.wrox.chapter5.controller.QuitCommand extends SimpleCommand {

      private var fscommand2:Function;

      public function execute(note:INotification):Void {
            fscommand2("Quit");
      }
}
```

Using execute()

In Listing 5-6 you see how the `QuitCommand` extends the `SimpleCommand`, and that it contains the `execute()` method. Consider when the `QUIT` notification is sent. What happens here is that the `Controller` iterates through the `commandMap` array property to find the `Command` object which is *mapped* to the notification name. In this case it is the `QuitCommand`. The `Controller` then calls the `execute()` method in the `Command` it finds, and then passes a reference notification `note` to trigger the command.

Within the `execute()` method of a concrete `SimpleCommand` object there are a number possibilities for your application. You can retrieve or manipulate data in the *model* via a proxy, or create or change components on the *view* via a Mediator, all with the option of passing a `Notification` object to each destination. In the `QuitCommand` you see the `fscommand2("Quit")` function is invoked, and the `note` is neither required nor utilized.

You extend the `MacroCommand` when you want to *execute* a number of `SimpleCommand` objects sequentially. This is handy, as it means any number of commands can be invoked without the need to fire off numerous notifications.

The `StartupCommand` is a concrete implementation of the `MacroCommand`.

Using initializeMacroCommand()

In stark contrast to creating a `MacroCommand` you simply extend the base class and then override the `initializeMacroCommand()` method. This should contain references to each `SimpleCommand` you want to *execute*. Passing the command as parameter to the `addSubCommand()` method (see Listing 5-9).

Creating the StartupCommand

Like with `ApplicationFacade`, the `StartupCommand` is frequently used in PureMVC based applications. Each of the applications found in this book use the `StartupCommand` to prep the application.

In creating the `StartupCommand` you'll reference two new commands not previously mentioned: `ModelPrepCommand` and `ViewPrepCommand`. You will create these commands later on in the chapter, but first let's look at the `StartupCommand`.

www.it-ebooks.info

1. First you need to import and extend the `MacroCommand` class (see Listing 5-7).



LISTING 5-7: Extending MacroCommand

```
import org.puremvc.as2.patterns.command.MacroCommand;

class com.wrox.chapter5.controller.StartupCommand extends MacroCommand {

}
```

2. Next ensure you have imported each of the `Command`'s you want to initialize in `StartupCommand`; these are your subcommands (see Listing 5-8).



LISTING 5-8: Importing ModelPrepCommand and ViewPrepCommand in StartupCommand

```
import com.wrox.chapter5.controller.ModelPrepCommand;
import com.wrox.chapter5.controller.ViewPrepCommand;

import org.puremvc.as2.patterns.command.MacroCommand;

class com.wrox.chapter5.controller.StartupCommand extends MacroCommand {

}
```

3. Finally, override the `initializeMacroCommand()` method. Then add the subcommands through `addSubCommand()`, supplying new instances of each `Command` (see Listing 5-9).



LISTING 5-9: Overriding the initializeMacroCommand method in StartupCommand

```
import com.wrox.chapter5.controller.ModelPrepCommand;
import com.wrox.chapter5.controller.ViewPrepCommand;

import org.puremvc.as2.patterns.command.MacroCommand;

class com.wrox.chapter5.controller.StartupCommand extends MacroCommand {
      private function initializeMacroCommand():Void {
            addSubCommand(new ModelPrepCommand());
            addSubCommand(new ViewPrepCommand());
      }
}
```



The order in which each addSubCommand() method is listed is quite significant as it determines the sequence in which Command is executed.

www.it-ebooks.info

Registering Commands with an Application

In order to invoke your concrete `Command` classes, they must be registered with the `Facade` using the `registerCommand()` method. This is usually done through the concrete `Facade`. Each part of the framework that extends the `Notifier` class can reference the `facade` instance and call its `registerCommand()` method.

It is through the `registerCommand()` method that the `Controller` saves a reference to a concrete `Command` object and the notification name in a local `commandMap` array property ready for notification.

If you have been following from earlier, Listing 5-3 showed you how to override the `initializeController()` method in `ApplicationFacade`. This is also where your commands are registered with the `Facade` (see Listing 5-10).



LISTING 5-10: Registering the StartupCommand and QuitCommand in ApplicationFacade

```
public static function getInstance():ApplicationFacade {
      if (instance == null) {
            instance = new ApplicationFacade();
      }

      return ApplicationFacade(instance);
}

private function initializeController():Void {
      super.initializeController();
      registerCommand(STARTUP, StartupCommand);
      registerCommand(QUIT, QuitCommand);
}
```



The `ApplicationFacade` *covered so far forms part of an example application, and in actual fact you could pretty much use this for the basis of every mobile application you wanted to create using PureMVC. Later in the chapter, at Listing 5-20, you will pick up from this point in the* `ApplicationFacade`*, where you tackle building the example PureMVC application.*

Having completed the discussion on *Commands and the Controller*, let's now turn our focus to the *Mediators and the View.*

Mediators And The View

The mediators facilitate communication between the view components and the rest of the PureMVC framework.

www.it-ebooks.info

The PureMVC framework provides a base class for concrete implementations of the `Mediator`. You'll find the class in the mediator package: `org/puremvc/as2/patterns/mediator/Mediator`.

The ViewPrepCommand

In the previous section you briefly touched on the `ViewPrepCommand`. While this is actually a command, it plays a significant role for in the model. This command is frequently defined in PureMVC applications to register each `Mediator` required for an application at startup (see Listing 5-11).



LISTING 5-11: An example ViewPrepCommand

```
import com.wrox.chapter5.view.AppMediator;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter5.controller.ViewPrepCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            var app:MovieClip = MovieClip(note.getBody());
            facade.registerMediator(new AppMediator(app));
      }
}
```

You will see variations of the `ViewPrepCommand` throughout the rest of the book, as you cover different *views*. The `ViewPrepCommand` in Listing 5-11 is used for the example application at the end of this chapter. It registers one concrete `Mediator` with the framework called `AppMediator`. You'll be taking a look at this mediator shortly, but before you do let's take an in-depth look at the logic behind the `registerMediator()` method.

Registering Mediators with an Application

As with the concrete `Command` you need to register concrete `Mediator`s using the `registerMediator()` method on the `Facade` instance. In AS2 the implementation `registerMediator()` requires that you create a new `Mediator` object. Listing 5-11 shows you the `execute()` method for the `ViewPrepCommand` and the `registerMediator()` method being called in the singleton instance of the `Facade`.

When the `registerMediator()` method is called the `View` saves a reference to the `Mediator` in its `mediatorMap Array` property. For this it requires the `NAME` property of the mediator needs to be declared as a `public static` property of the concrete `Mediator`. By convention the value of `NAME` should be the mediator class name.

www.it-ebooks.info




```
public static var NAME:String = "AppMediator";
```

code snippet com/wrox/chapter5/view/AppMediator.as

The `View` calls the `getMediatorName()` method on each mediator it registers to retrieve the `mediatorName` property, and then retrieves a list of notifications the `Mediator` object has listed as having an interest in, invoking its `listNotificationInterests()` method.

Using listNotificationInterests()

When you create your concrete `Mediator` classes you need to consider how they need to respond to changes in your application, and if they need to observe any of notifications that are being broadcast. In each mediator you override the `listNotificationInterests()` method and place each notification name you want the mediator to handle into an array, which is returned when the method is invoked. The `view` then iterates through the list. `AppMediator` is interested in three notifications: `SECTION_UPDATED`, `GET_NEXT_SECTION` and `GET_PREVIOUS_SECTION` (see Listing 5-12). There will be more on these later.



LISTING 5-12: The listNotificationInterests method for the AppMediator

```
public function listNotificationInterests():Array {
      return [ ApplicationFacade.SECTION_UPDATED,
               ApplicationFacade.GET_NEXT_SECTION,
               ApplicationFacade.GET_PREVIOUS_SECTION ];
}
```

When the `listNotificationInterests()` method is called the `View` assigns an `Observer` object to each notification interest, assigning the `Notification` object to the `handleNotification()` method on the `Mediator` object. This is all handled by the PureMVC framework.

Using handleNotification()

So once you've decided what notifications you want your mediator to respond to, you need to create handlers for each notification. In your concrete `Mediator`, you override the `handleNotification()` method and check for the correct notification. This is done by implementing a switch statement on the `getName()` method on the `Notification` object (see Listing 5-13).



LISTING 5-13: The handleNotification method for the AppMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SECTION_UPDATED:
                  var sectionVO:SectionVO = sectionVO(note.getBody());
                  titleText.sectionName = sectionVO.name;
```

continues

www.it-ebooks.info

LISTING 5-13 *(continued)*

```
                descriptionText.sectionDescription = sectionVO.description;
            break;
        }
    }
```

When `registerMediator()` is called in the `ViewPrepCommand`, the `View` initiates the `Mediator` by calling its `onRegister()` method. In your concrete `Mediator` you can override this method to define what happens as soon as the `Mediator` is registered by the `Facade`.

You'll take a look at each one of these concepts while creating `AppMediator`.

Creating AppMediator

The name `AppMediator` is really just a non-descript name for an example mediator utilized in the example PureMVC framework application used at the end of the chapter to simply display content. It basically consists of a few components. You will come across many more mediators throughout the remainder of the book. Here you can run through the first few aspects of creating the `AppMediator` used at the end of this chapter, some of which you have already touched upon. You can create this file just like the `ApplicationFacade`, with the intention of returning to it later in the chapter for completing the *chapter5_PureMVC* example, or you can simply follow the steps just to grasp how mediators are constructed.

1. First, `AppMediator` extends `Mediator` and implements `IMediator`; you will need to ensure both files are imported (see Listing 5-14).



LISTING 5-14: Creating the AppMediator class

```
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter5.view.AppMediator extends Mediator implements IMediator
{

}
```

2. Next define the `NAME` property and constructor for the mediator, calling the `super()` constructor, and passing references to `NAME` and the `viewComponent` (see Listing 5-15).



LISTING 5-15: Defininining the name and constructor for the AppMediator

```
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter5.view.AppMediator extends Mediator implements IMediator {

      public static var NAME:String = "AppMediator";
```

www.it-ebooks.info

```
        public function AppMediator(viewComponent:Object) {
              super(NAME, viewComponent);
        }
}
```

The constructor method in the mediator takes an object, which if you recall from earlier, is a reference to the application.

3. Next you need to create the getter method to return the type of the `viewComponent`, passed to the mediator. Here the return type is a `MovieClip` (see Listing 5-16).



LISTING 5-16: Casting the viewComponent to its actual type

```
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter5.view.AppMediator extends Mediator implements IMediator {

      public static var NAME:String = "AppMediator";

      public function AppMediator(viewComponent:Object){
            super(NAME, viewComponent);
      }

      public function get _component():MovieClip {
            return MovieClip(viewComponent);
      }
}
```

Here you've defined the `_component()` method and cast the `viewComponent` object to its actual data type. This is an important step as you now have a means whereby you can add components to the stage of the application, by either creating them dynamically or attaching them from the library in your application's `.fla` file.

4. Next create two MovieClip based components via the `onRegister()` method. The focus here is the `attachMovie()` reference, which adds two MovieClip components called `title` and `description` from the Library of a Flash movie to `AppMediator`. The `attachMovie()` call uses the `_component` getter method you defined in the previous step, and demonstrates how to initialize components in a mediator (see Listing 5-17). Not all components however need to the return type of a `MovieClip`.



LISTING 5-17: Using onRegister to add components to the application via the AppMediator

```
public static var NAME:String = "AppMediator";

private var titleTxt:MovieClip;
private var descriptionTxt:MovieClip;
```

continues

www.it-ebooks.info

LISTING 5-17 *(continued)*

```
public function AppMediator(viewComponent:Object) {
      super(NAME, viewComponent);
}

public function onRegister():Void {
      titleTxt = _component.attachMovie("title", "t", 1);
      descriptionTxt = _component.attachMovie("description", "d", 2)
;}

public function get _component():MovieClip {
      return MovieClip(viewComponent);
}
```

5. Next create the `listNotificationInterests()` and `handleNotification()` methods. Here you need to define the notifications `AppMediator` is interested in. Modify the `listNotificationInterests()` method to listen to a notification name called `SECTION_UPDATED`. The mediator will be responsible for updating text within the `titleTxt` and `descriptionTxt` components. Each has a dynamic text field which exposes a variable for assigning content. Modify the `handleNotification()` method so that when the `SECTION_UPDATED` notification has been sent, the `titleTxt` and `descriptionTxt` components are updated (see Listing 5-18).



LISTING 5-18: The listNotificationInterests and handleNotification methods for AppMediator

```
public function onRegister():Void {
      titleTxt = _component.attachMovie("title", "t", 1);
      descriptionTxt = _component.attachMovie("description", "d", 2);
}

public function listNotificationInterests():Array {
      return [ApplicationFacade.SECTION_UPDATED];
}

public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SECTION_UPDATED:
                  var sectionVO:SectionVO = SectionVO(note.getBody());
                  titleTxt.sectionName = sectionVO.name;
                  descriptionTxt.sectionDescription = sectionVO.description;
            break;
      }
}
```

The `SectionVO` highlighted in Listing 5-18 is a value object which will be covered later shortly; all you need to be aware of here is that the value return contains two string based properties which is assigned to the `sectionName` variable of the `titleTxt` component, and `sectionDescription` of the `descriptionTxt` component. The `SectionVO` essentially contains data that is stored via the model of the example PureMVC application created in

www.it-ebooks.info

this chapter, which leads us into discussion on *Proxies and the Model*. The remainder of `AppMediator` is completed towards the end of this chapter.

Proxies and the Model

Proxies are used to manage an application's data model within PureMVC. For your applications you create a proxy by extending the base class provided by the PureMVC framework. You'll find the base class for implementing a proxy in the `proxy` package: `org/puremvc/as2/patterns/proxy/Proxy.as`.

Throughout the book there will be numerous implementations of proxies, which you will be able to reuse for applications.

Registering Proxies with an Application

So far you've covered registering commands and mediators with an application. The `Proxy` classes are registered through the `registerProxy()` method, usually when a command is executed. The `execute()` method of the `ModelPropCommand`, which you'll explore later, is a good example of the registering proxy classes. Listing 5-19 shows how the `SectionProxy` is registered with a PureMVC application.



LISTING 5-19: Registering the SectionProxy in ModelPrepCommand

```
public function execute(note:INotification):Void {
      facade.registerProxy(new SectionProxy());
}
```

As with the `Command` and `Mediator` you need to register your concrete `Proxy` through the `facade.registerProxy()` method. When this method is invoked, the *Model* saves a reference to the `Proxy` object in its `proxyMap` property. It then calls the `onRegister()` method in the `Proxy` object; this should all sound familiar.

Inside your `Proxy` you could override `onRegister()` and implement a reference to retrieve or set your data, or you can create a method which needs to be called directly.

Using setData()

Each `Proxy` has a `data` object property, which can be saved via calling the `setData()` method. As with the `Mediator`, the `Proxy` is also able to send `Notification` objects via the use of the `sendNotification()`. Proxies should only actually send notifications.

Using Retrieve and Remove Methods

The `Facade` provides methods for retrieving parts of the model and view via its `retrieveMediator()` and `retrieveProxy()` method. You reference the `Mediator` and `Proxy` through its `NAME` property which is static. The `Facade` also provides methods for removing registered mediators, commands, or proxies as part of a cleanup process including: `removeMediator()`, `removeCommand()`, and `removeProxy()`.

www.it-ebooks.info



The retrieve and remove methods should **only** *be called by the commands.*

You've now covered each key *actor* within the PureMVC framework and their respective properties. Phew! While you've now got a good grounding in PureMVC there's nothing like putting theory into practice . . .

BUILDING A PUREMVC EXAMPLE

In this section you'll be creating a basic Flash enabled mobile device application using the PureMVC framework. The example will simply display a summary from the sections of this chapter's topics, by pressing left or right on the device. The aim is to give you a complete PureMVC example utilizing *Commands*, *Mediators*, *Proxies*, and the *ApplicationFacade*.

Using PureMVC, it's good practice to create a conceptual diagram for your Flash Lite application, as it will often highlight before coding whether your application is headed in the right direction. It makes it easier to break down the conceptual diagram into each core responsibility.

Figure 5-3 shows the architecture for the example application, and this is based off of the conceptual diagram.

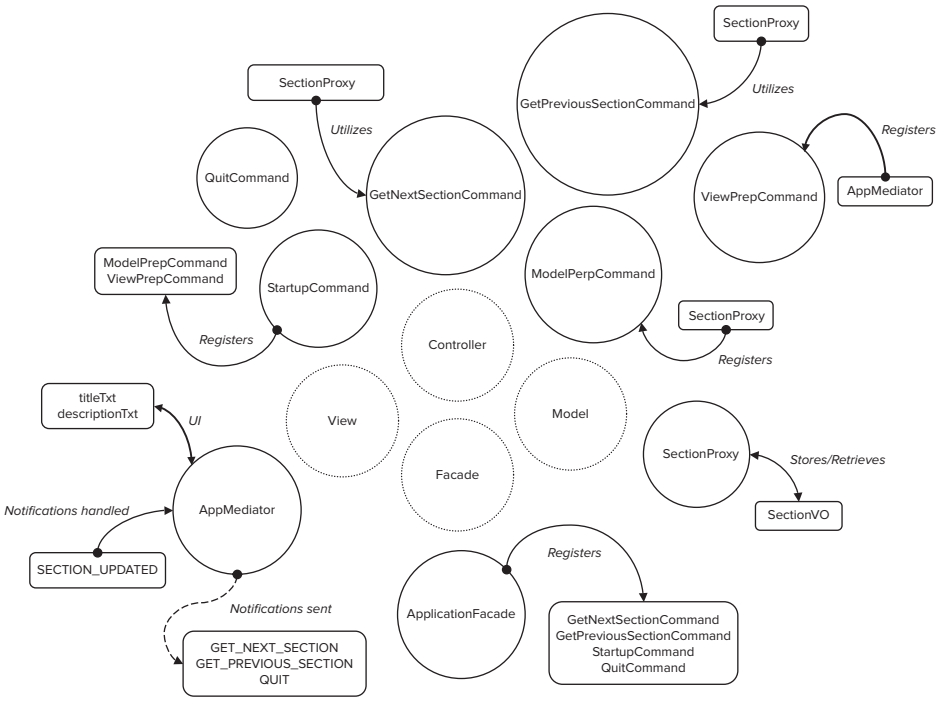


FIGURE 5-3

www.it-ebooks.info

You will see from Figure 5-3 that we will create ten classes for the application based on the existing PureMVC framework: one concrete Facade, one mediator, six commands, one proxy, and a value object to store our data.

The `chapter5_PureMVC.fla` file accompanying this chapter has the layout and library assets all setup and ready for Flash CS4. The Project panel shown in Figure 5-4 indicates the file system layout.

Figure 5-4 shows the Projects Panel containing the files for the task.

If you have been following you now should be ready to follow the steps for completing your first PureMVC based Flash Lite application. By this stage you should have a `StartupCommand`, `QuitCommand`, and `ViewPrepCommand` all done. You should also have the `AppMediator` and `ApplicationFacade` classes partially completed.



FIGURE 5-4

Creating ApplicationFacade Continued

If you've been following from earlier, up to Listing 5-10 showed you how to complete the first parts of `ApplicationFacade`.

1. Return to the `ApplicationFacade` and add a further three notification names: `GET_NEXT_SECTION`, `GET_PREVIOUS_SECTION`, and `SECTION_UPDATED` (see Listing 5-20).



Available for download on Wrox.com

LISTING 5-20: Adding new notification names to the ApplicationFacade

```
public static var STARTUP:String = "startup";
public static var GET_NEXT_SECTION:String = "getNextSection";
public static var GET_PREVIOUS_SECTION:String = "getPreviousSection";
public static var SECTION_UPDATED:String = "sectionUpdated";
public static var QUIT:String = "quit";

public static function getInstance():ApplicationFacade {
      if (instance == null) {
            instance = new ApplicationFacade();
      }

      return ApplicationFacade(instance);
}

private function initializeController():Void {
      super.initializeController();
}
```

continues

www.it-ebooks.info

LISTING 5-20 *(continued)*

```
public function startup(app:MovieClip):Void {
      sendNotification(STARTUP, app);
}
```

2. Modify the `initializeController()` function to register two new commands called `GetNextSectionCommand` and the `GetPreviousSectionCommand` (see Listing 5-21).



LISTING 5-21: Registering two additional commands in ApplicationFacade

```
public static function getInstance():ApplicationFacade {
      if (instance == null) {
            instance = new ApplicationFacade();
      }
            return ApplicationFacade(instance);
}

private function initializeController():Void {
      super.initializeController();
      registerCommand(STARTUP, StartupCommand);
      registerCommand(QUIT, QuitCommand);
      registerCommand(GET_NEXT_SECTION, GetNextSectionCommand);
      registerCommand(GET_PREVIOUS_SECTION, GetPreviousSectionCommand);
}
```

This completes the `ApplicationFacade`; the new commands mentioned in the last step will be covered shortly. First let's take a look at the model for the app.

Creating a Model for the Example App

In this section you cover the data object used in the application called `SectionVO`; if you remember earlier this was mentioned briefly in the creation of `AppMediator`. The second part to creating the model is the proxy for the application, called `SectionProxy`. Again you can follow the steps to create the classes or just follow the text for a further understanding of *Proxies and the Model*.

Using Value Objects and Proxies to Store and Manage Data

Earlier you saw how the `AppMediator` referenced the `SectionVO` objects to assign values to the `titleTxt` and `descriptionTxt` components. This was actually retrieved through the notification, which was sent by the `SectionProxy`.

SectionVO.as

The `SectionVO` is a value object which simply holds two public properties, `name` and `description`, and is a data type which can be easily referenced in the code base (see Listing 5-22).

www.it-ebooks.info



LISTING 5-22: The SectionVO class

```
class com.wrox.chapter5.model.vo.SectionVO {

      public var name:String;
      public var description:String;

      public function SectionVO() {

      }
}
```

SectionProxy.as

For the `SectionProxy` class you are simply storing data which can be retrieved by the application. The class has four custom methods: `createData()`, `setSection()`, `previousSection()`, and `nextSection()`. In addition, the `onRegister()` method needs to be defined.

1. Begin to create the `SectionProxy` class by extending the `Proxy` and implementing `IProxy` (see Listing 5-23).



LISTING 5-23: Creating the SectionProxy

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter5.model.SectionProxy extends Proxy implements IProxy {

      public static var NAME:String = "SectionProxy";

      public function SectionProxy(data:Object) {
           super(NAME, data);
      }
}
```

2. Next add two private variables called `sections` and `index`. The `sections` variable is an array to store each of the `SectionVO` data objects which need to be referenced in the application, whereas the `index` variable is a number to keep track of the currently selected `SectionVO` object. The two are used in conjuction with each other to return the correct text to display in the app. The `SectionVO` class will need to be imported here, along with `ApplicationFacade`, which needs to be imported so that the proxy can send notifications (see Listing 5-24).



LISTING 5-24: Adding private variables to SectionProxy

```
import com.wrox.chapter5.ApplicationFacade;
import com.wrox.chapter5.model.vo.SectionVO;

import org.puremvc.as2.interfaces.IProxy;
```

continues

www.it-ebooks.info

LISTING 5-24 *(continued)*

```
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter5.model.SectionProxy extends Proxy implements IProxy {

      public static var NAME:String = "SectionProxy";

      private var sections:Array;
      private var index:Number = 0;

      public function SectionProxy(data:Object) {
           super(NAME, data);
      }
}
```

3. Next create the public `setSection()` method. The method has one argument, the `value`, a number which basically sets the value for the `index` property. Set this to 0. Here you'll need a conditional statement to check whether the number is valid and less than the number of items contained in the `sections` array. Once the `index` has been set it's important to note that the `SECTION_UPDATE` notification needs to be sent (see Listing 5-25).



LISTING 5-25: Defining the setSection method for SectionProxy

```
public function setSection(value:Number):Void {

      if (value > sections.length) {
            index = 0;
      } else {
            index = value;
      }

      sendNotification(ApplicationFacade.SECTION_UPDATED, sections[index]);
}
```

4. Next create the `createData()` method. As the name suggests you are essentially creating the data for the application. Here the detail for each section of Chapter 5 is summarized and the `SectionVO` class is utilized. The `name` and `description` properties of the value object are defined for five different sections. Each `SectionVO` instance needs to be added to the `sections` array, and then the `setSection()` method needs to be called with the current value for `index` supplied as the parameter (see Listing 5-26).



LISTING 5-26: Defining the createData method for SectionProxy

```
private function createData():Void {

      var section1:SectionVO = new SectionVO();
      section1.name = "CHAPTER 5: ";
      section1.description = "PureMVC ActionScript 2.0 Framework. ";

      var section2:SectionVO = new SectionVO();
```

www.it-ebooks.info

```
      section2.name = "1. Design Patterns And Development Frameworks ";
      section2.description = "In this section we go over the Design Patterns used
in PureMVC: \n \na. Time for a development framework \nb. Design Patterns used in
PureMVC";

      var section3:SectionVO = new SectionVO();
      section3.name = "2. PureMVC Key Concepts Explained ";
      section3.description = "In this section we go over the key concepts of
PureMVC: \n \na. The Facade \nb. Notifications and the Observer \nc. Commands and
the Controller \nd. Mediators and the View \ne. Proxies and the Model";

      var section4:SectionVO = new SectionVO();
      section4.name = "3. Building a PureMVC Example ";
      section4.description = "In this section you create a Flash enabled mobile
application from scratch using the PureMVC framework.";

      var section5:SectionVO = new SectionVO();
      section5.name = "4. Summary";
      section5.description = "End of chapter 5 and summary.";

      sections = [section1, section2, section3, section4, section5];

      setSection(index);
}
```

5. Next create the `nextSection()` and `previousSection()` methods. As these names indicate, the methods are simply used to retrieve the next and previous sections defined from the data set via `createData()`. Here the `index` variable needs to be manipulated so that when the `nextSection()` method is called the `index` is incremented, while when the `previousSection()` is called the reverse is true. In both instances the `SECTION_UPDATED` notification needs to be sent with the selected section in the notification body; this is done through the `sections[index]` (see Listing 5-27).



LISTING 5-27: Defining the nextSection and previousSection methods for the SectionProxy

```
public function nextSection():Void {

      index++;

      if (index >= sections.length) {
            index = 0;
      }

      sendNotification(ApplicationFacade.SECTION_UPDATED, sections[index]);
}

public function previousSection():Void {

      index--;
```

continues

www.it-ebooks.info

LISTING 5-27 *(continued)*

```
        if (index <= -1) {
              index = 0;
        }

        sendNotification(ApplicationFacade.SECTION_UPDATED, sections[index]);
}
```

6. For the final step in creating the `SectionProxy`, ensure that the `onRegister()` method for the `SectionProxy` is overridden to call the `createData()` method. This will ensure that when the proxy is registered in `ModelPrepCommand` the data for the application is created immediately (see Listing 5-28).



LISTING 5-28: Overriding the onRegister method in the SectionProxy

```
public function onRegister():Void {
      createData();
}
```

This now completes the model portion of the application. Following on from here the pieces are in place to add *controller* elements for the example application.

Creating the Example App's Controller

There are several commands used in the example application, some of which you have already covered, including `StartupCommand`, `QuitCommand`, and `ViewPrepCommand`. The remaining commands are `ModelPrepCommand`, `GetNextSectionCommand`, and `GetPreviousSectionCommand`. You'll take a look at each of these in turn.

Registering Proxies with the Application

Earlier you covered registering proxies, mediators and commands with the application so this section should be familiar.

ModelPrepCommand.as

The `ModelPrepCommand`, which extends the `SimpleCommand`, needs to register the `SectionProxy` with the application through the `execute()` method (see Listing 5-29).



LISTING 5-29: Registering the SectionProxy in ModelPrepCommand

```
import com.wrox.chapter5.model.*;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;
```

www.it-ebooks.info

```
class com.wrox.chapter5.controller.ModelPrepCommand extends SimpleCommand {

      public function execute(note:INotification):Void {
            facade.registerProxy(new SectionProxy());
      }
}
```

Using Commands to Navigate Through Data

As mentioned there are two commands to navigate through data, the `GetNextSectionCommand` and the `GetPreviousSectionCommand`. The commands are very similar: both extend `SimpleCommand`, and both retrieve the `SectionProxy` defined by the model. The commands demonstrate a common approach to utilizing data in PureMVC.

GetNextSectionCommand.as

The `GetNextSectionCommand` needs to call the `nextSection()` method defined in the `SectionProxy` (see Listing 5-30).



LISTING 5-30: The GetNextSectionCommand

```
import com.wrox.chapter5.model.SectionProxy;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter5.controller.GetNextSectionCommand extends SimpleCommand
implements ICommand {

      public function execute(note:INotification):Void {

            var sectionProxy:SectionProxy = SectionProxy(facade.retrieveProxy(Sectio
nProxy.NAME));
            sectionProxy.nextSection();
      }
}
```

GetPreviousSectionCommand.as

The `GetPreviousSectionCommand` needs to call the `previousSection()` method defined in the `SectionProxy` (see Listing 5-31).



LISTING 5-31: The GetPreviousSectionCommand

```
import com.wrox.chapter5.model.vo.SectionProxy;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.command.*;
```

continues

www.it-ebooks.info

LISTING 5-31 *(continued)*

```
class com.wrox.chapter5.controller.GetPreviousSectionCommand extends SimpleCommand
implements ICommand {

      public function execute(note:INotification):Void {

            var sectionProxy:SectionProxy = SectionProxy(facade.retrieveProxy(Sectio
nProxy.NAME ));
            sectionProxy.previousSection();

      }
}
```

Each of the commands required for the example application are now completed. With the data model defined also, the remaining aspect is to cover the view aspect and the mediators used there.

Exploring the View

Earlier in the chapter you began to create the `AppMediator`. This is actually the only mediator used in the application and Listing 5-18 was where you had just defined the `listNotificationInterests()` and `handleNotification()` methods.

Creating AppMediator Continued

1. Returning to the `AppMediator`, ensure that the alignment of the `titleTxt` and `descriptionTxt` components are set in the `onRegister()` method (see Listing 5-32).



LISTING 5-32: Aligning components in the onRegister method for AppMediator

```
public function onRegister():Void {
      titleTxt = _component.attachMovie("title", "t", 1);
      titleTxt._x = 5;
      titleTxt._y = 2;

      descriptionTxt = _component.attachMovie("description", "d", 2);
      descriptionTxt._x = 5;
      descriptionTxt._y = 28;
}
```

2. Next you need to define how the `titleTxt` and `descriptionTxt` components will be updated. The approach here is to use key presses to cycle through the section data in the proxy. So first assign the key presses to the `GET_PREVIOUS_SECTION` and `GET_NEXT_SECTION` notifications in the `onKeyDown()` method (see Listing 5-33).

www.it-ebooks.info



LISTING 5-33: The onKeyDown method

```
public function onKeyDown():Void {

      switch (Key.getCode()) {
            case Key.UP:
                  sendNotification(ApplicationFacade.GET_PREVIOUS_SECTION);
            break;
            case Key.DOWN:
                  sendNotification(ApplicationFacade. GET_NEXT_SECTION);
            break;
            case Key.LEFT:
                  sendNotification(ApplicationFacade.GET_PREVIOUS_SECTION);
            break;
            case Key.RIGHT:
                  sendNotification(ApplicationFacade.GET_NEXT_SECTION);
            break;
      }
}
```

The `onKeyDown()` method should capture the up, left, right and down key event presses.

3. Next update the `onRegister()` method so it includes the `addListener()` method to register the `AppMediator` as the event listener for the `Key` object. This ensures the `onKeyDown` event is handled by the `onKeyDown()` method you defined in the last step (see Listing 5-34).



LISTING 5-34: Initializing the Key listener

```
public function onRegister():Void {
      titleTxt = _component.attachMovie("title", "t", 1);
      titleTxt._x = 5;
      titleTxt._y = 2;

      descriptionTxt = _component.attachMovie("description", "d", 2);
      descriptionTxt._x = 5;
      descriptionTxt._y = 28;
      Key.addListener(this);
}
```

4. Finally, ensure a notification to quit the application can be invoked. This should be handled by the `ExtendedKey.SOFT1` (see Listing 5-35).



LISTING 5-35: Assigning a key to quit the application

```
public function onKeyDown():Void {

      switch (Key.getCode()) {
```

continues

www.it-ebooks.info

LISTING 5-35 *(continued)*

```
                case ExtendedKey.SOFT1:
                      sendNotification(ApplicationFacade.QUIT);
                break;
                case Key.UP:
                      sendNotification(ApplicationFacade.GET_PREVIOUS_SECTION);
                break;
                case Key.DOWN:
                      sendNotification(ApplicationFacade.GET_NEXT_SECTION);
                break;
                case Key.LEFT:
                      sendNotification(ApplicationFacade.GET_PREVIOUS_SECTION);
                break;
                case Key.RIGHT:
                      sendNotification(ApplicationFacade.GET_NEXT_SECTION);
                break;
          }
    }
```

You now have covered each of the elements for the example PureMVC application.

Viewing the Application in Device Central

1. Within Flash CS4, open the `chapter5_PureMVC.fla`. On the first frame of the timeline ensure the lines which will kick start the application are present (see Listing 5-36).



LISTING 5-36: Initializing the example application in on the timeline

```
import com.wrox.chapter5.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
app.startup(this);
```

2. Still within Flash CS4, you need to test run your application: select Control ➪ Test Movie . . ., Device Central should launch allowing you to interact with the application by pressing the keys on the D-pad. You can also click the Test Project button in the Project panel.

Figure 5-5 shows a screen of the completed example application on the Nokia E71.

The PureMVC example application is now complete!



FIGURE 5-5

SUMMARY

In this chapter, you've learned all aspects of PureMVC and should have a solid base for developing well-structured Flash enabled mobile device applications using the framework. You've also explored the

www.it-ebooks.info

key concepts of PureMVC, reviewing the roles of the key Actors. You should know your Facade from your Command and your Mediator from your Proxy.

You've learned how to startup and quit applications. Hopefully, you're starting to realize the benefits of having a development framework for Flash Lite.

In the next chapter, you'll use the framework to create a more involved example — a TV listings application — where you'll learn about more conceptual elements.

For more comprehensive coverage of PureMVC, visit the official site at `http://www.puremvc.org`.

www.it-ebooks.info

6

Creating a TV Listings Application

WHAT'S IN THIS CHAPTER?

- Utilizing XML
- Security considerations when loading data
- Using PureMVC to build a TV Listings application

So far you've been given an overview of Flash Lite, object-oriented programming, UI components, and the PureMVC framework. In this chapter you will build on the knowledge gained so far in the opening chapters and explore a TV listings application.

You'll explore an API developed by TV Genius (`http://www.tvgenius.net`) founded by Tom Weiss, who provides services for integrating with their innovative TV listings search engine.

THE CONCEPT

In recent years the development of YouTube, BBC iPlayer, Sky Player, and other TV-related Internet services have changed the way in which we consume television and video clips. When I developed a Flash Lite Mobile TV Guide in 2005, it was a proof of concept, which was loosely based on what I saw in the *Radio Times* magazine. In the application you were able to select a day of the week, pick a channel, and then view exactly what was on for that day, and if a title looked interesting, you were able to read a little description. The application

www.it-ebooks.info

is very well suited to mobile devices because it means that you are able to view TV listings wherever you are. With the popularity of TV on demand via the web, gaming consoles, and television sets, mobile applications that feed us the information to make informed choices on *what* to watch, and *when*, will certainly evolve in the coming years.

In this chapter you'll explore the build for a mobile device application that delivers TV listings based on the keyword search terms that a user provides, using the TV Genius Search API.

TV GENIUS SEARCH ENGINE API

The TV Genius search engine provides you with details of shows that are available to watch on television or online. You provide a search term and it will return you a list of results, with show titles, descriptions, related shows, linked episodes.

Using the TV Genius search engine API consists of a simple call to an `XML` interface over HTTP:

```
http://listings.tvgenius.co.uk/feeds/search.xml
```

Here is an example of a `URL` request to the TV Genius service with parameters:

```
http://listings.tvgenius.co.uk/feeds/search.xml?tvgBroadcaster=OnlineTV&tvgSearch
Phrase=formula 1&tvgResultsPerPage=5&tvgPageNumber=1&tvgReferer=jgorganic
```

In the example shown, there are a number of name/value parameters that have been set in order to return specific results from TV Genius; let's take a closer look at these.

Mandatory Search Parameters

Each call to the service requires you to send several mandatory parameters, which must be set; these include:

- **Referrer —** A unqiue code provided by TV Genius that authenticates the use of the service. The referer is defined by the `tvgReferer` parameter. This is set to `jgorganic` for the purpose of the TV Listings application and walkthrough.
- **Broadcaster —** An option that determines which channels the search will filter by. The broadcaster is defined by the `tvgBroadcaster` parameter. This will be set to `OnlineTV`.



To retrieve your own unique referrer key, please email support@tvgenius.co.uk.

Optional Search Parameters

There are also optional parameters you can supply with your search query; these include:

www.it-ebooks.info

- **Search phrase** — A value representing the search term. The search phrase is defined by the `tvgSearchPhrase` parameter string.
- **Trim description** — A value to set the number of characters returned in descriptions. The trim is defined by the `tvgTrimDescription` parameter.
- **Results per page** — A value to define the number of results returned for each request. The number of results to return is defined by the `tvgResultsPerPage` parameter.
- **Page number** — A value used to set which page of results should be returned, as defined by `tvgResultsPerPage`. The page number is defined by the `tvgPageNumber` parameter.

Response Format for an API Call

Now that you've had a brief view of the parameters used in the calls to the TV Genius API, here's a code snippet detailing an example of the response from the server, showing the XML format and how the results can be returned:




```
<almostRss>
      <title>TV Genius found 1 to 5 of 67 shows matching "Star Wars" on all
Web TV channels</title>
      <channel>
            <item>
                  <title>Star Wars: The Clone Wars</title>
                  <description>Star Wars: The Clone Wars: Time to return once more
to a galaxy far, far away as Cartoon Network's punk animation Clone Wars gets the
CGI treatment for a stellar adventure packed with epic battles and faces old and
new. Star Wars' first animated feature depicts the unfolding tragedy of the Galactic
Republic as it slips further under the sway of the Separatists and their fearsome
droid army. Director Dave Filoni knows what makes Star Wars fizz, and for those
still smarting from the prequels there is enough of the old magic here to make
their lightsabers hum. Related Movies: By Genre Starring Samuel L Jackson Starring
Tom Kane Starring Matt Lanter Starring Christopher Lee Directed by Dave Filoni
Animation, Family, Sci-fi</description>
                  <availability>Available now on Sky Anytime Movies P-S for 363
days</availability>
            </item>
      </channel>
</almostRss>
```

code snippet almostRss.xml

Utilizing the details of an `XML` file, and how you can display the information in your mobile applications, will be discussed later in this chapter. For now take note of the `<title />`, `<description />`, and `<availability />` nodes of the result, which you will be parsing.

Next, you'll go over the criteria for the application.

www.it-ebooks.info

FUNCTIONAL REQUIREMENTS

In this section you will take a look at some examples of functional requirements used to define the features of the TV Listings application.

The functional requirements simply set criteria that the application needs to adhere to, and every set of functional requirements brings with it a few hidden challenges:

1. Allow the user to key text input and submit a search query:

   Here the challenge will be to capture a value entered into a dynamic text field by the user and then forward it in a query string to the TV Genius server.

2. Provide network request and connectivity status:

   Here the network requests and connectivity statuses classes, covered in Chapter 3, will need to be utilized to present network information to the user.

3. Allow the user to view the search results:

   Here the main challenge is handling the XML data format returned by a search query, a format that will need to be converted to something more readable. A method for iterating through the results needs to be implemented also in the search view.

4. Allow the user to view a description of each result:

   Here the user should be able to get more information about the particular show after the initial results have been retrieved.

5. Allow the user to perform multiple searches:

   Lastly, once the user has performed one search, he or she should be able to submit another. Hence a method for ensuring that users will always find their way back to the search view from the results section or the description needs to be implemented.

These are just a few examples of functional requirements for a mobile application. In the next section you'll take a look at how each of these requirements is implemented.

BUILDING THE APPLICATION

In this section you'll get an in-depth view of the code for the TV Listings application and gain further insight into the conceptual diagrams for the application, which are based on the PureMVC framework.

You'll go through each aspect of the classes created for the project, and along the way we'll point out areas that need particular attention, taking a look at the following:

- The `.fla` file
- Defining ApplicationFacade
- Creating a Model for the TV Listings application

www.it-ebooks.info

- Creating the TV Listings application's controller
- Exploring the view's mediators

The .fla File

Open the `chapter6_TV_Listings.fla` file in Flash CS4. In the Library panel for the application, you'll see that there are several assets, all of which are used in the application, and, on the first frame of the timeline, you'll find the script to kick start your application.

Defining ApplicationFacade

The `ApplicationFacade` for the TV Listings application has a very similar structure to that of the class covered in Chapter 5. The main differences, of course, are the notification names and the respective mappings to the commands (see Listing 6-1).



LISTING 6-1: The ApplicationFacade for TV Listings application

```
import com.wrox.chapter6.model.*;
import com.wrox.chapter6.view.*;
import com.wrox.chapter6.controller.*;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.facade.*;

class com.wrox.chapter6.ApplicationFacade extends Facade implements IFacade {
      public static var STARTUP:String = "startUp";
      public static var QUIT:String = "quit";
      public static var GET_NEXT_RESULT:String = "getNextResult";
      public static var GET_PREVIOUS_RESULT:String = "getPreviousResult";
      public static var SHOW_MAIN_MENU:String = "showMainMenu";
      public static var SHOW_SEARCH_PANEL:String = "ShowSearchPanelCommand";
      public static var SEARCH:String = "search";
      public static var DISPLAY_RESULTS:String = "displayResults";
      public static var VIEW_DETAILS:String = "viewDetails";

      public static function getInstance():ApplicationFacade {
            if (instance == null) {
                  instance = new ApplicationFacade();
            }
            return ApplicationFacade(instance);
      }

      private function initializeController():Void {
            super.initializeController();
            registerCommand(STARTUP, StartupCommand);
            registerCommand(SHOW_MAIN_MENU, ShowMainMenuCommand);
            registerCommand(QUIT, QuitCommand);
            registerCommand(SHOW_SEARCH_PANEL, ShowSearchPanelCommand);
            registerCommand(SEARCH, SearchCommand);
```

continues

www.it-ebooks.info

LISTING 6-1 *(continued)*

```
            registerCommand(GET_NEXT_RESULT, GetNextResultCommand);
            registerCommand(GET_PREVIOUS_RESULT, GetPreviousResultCommand);
            registerCommand(VIEW_DETAILS, ViewDetailsCommand);
      }

      public function startup(app:MovieClip):Void {
            sendNotification(STARTUP, app);
      }
}
```

Defining the Notification Names

There are several notification names defined in `ApplicationFacade`, which are registered within the private `initializeController()` method. Here, they are mapped to their respective commands:

- `STARTUP` — Mapped to the `StartupCommand`, and used to start the application.
- `GET_NEXT_RESULT` — Mapped to `GetNextResultCommand` and, when called, is responsible for returning the next search result.
- `GET_PREVIOUS_RESULT` — Mapped to `GetPreviousResultCommand` and returns the previous search result.
- `SHOW_MAIN_MENU` — Mapped to `ShowMainMenuCommand` and is responsible for displaying the main menu.
- `SHOW_SEARCH_PANEL` — Mapped to `ShowSearchCommand` and, when called, is responsible for displaying the search panel.
- `SEARCH` — Mapped to `SearchCommand` and, when called, is responsible for initializing the search.
- `DISPLAY_RESULTS` — Mapped to `DisplayResultsCommand` and, when called, is responsible for displaying the results from the server.
- `VIEW_DETAILS` — Mapped to `ViewDetailsCommand` and is responsible for presenting the details of a result for viewing.
- `QUIT` — Mapped to the `QuitCommand` and used to quit and close the application.

You'll cover each of these commands later in this chapter. Next up is creating the model.

Creating a Model for the TV Listings Application

The model will consist of three proxies: one for fscommands, called `FSCommandsProxy`; one for the search results, called `ResultsProxy`; and lastly one for handling *network requests* and *connectivity*, called `NetworkProxy`. Figure 6-1 shows an isolated conceptual diagram for the model, along with some of the core classes utilized.

www.it-ebooks.info

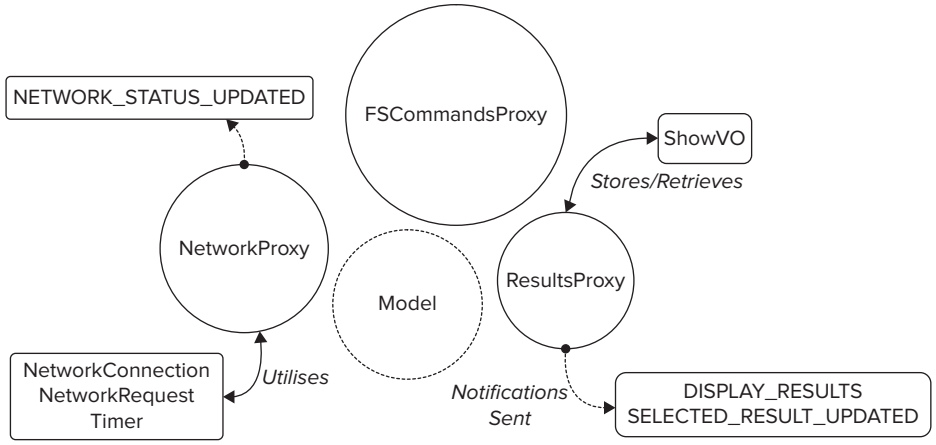


FIGURE 6-1

Here, you see there are `Proxy` classes utilized in the application:

- `FSCommandsProxy` — To provide default settings for the application.
- `ResultsProxy` — To retrieve and provide the TV listings' data results from the TV Genius server.
- `NetworkProxy` — To provide the network request, status, and connection data.

The Project panel, which is shown in Figure 6-2, illustrates the file system layout for the TV Listings application. Figure 6-2 shows each of the files under the model directory.



FIGURE 6-2

Before looking at the new proxy classes in detail, take a look at the value object that is used in the application, called `ShowVO`.

ShowVO

The `ShowVO` class is a value object that will represent a single result retrieved from the server. It has three properties: `title`, `description`, and `availability`. Each will be utilized in the application (see Listing 6-2).



LISTING 6-2: The ShowVO class

```
class com.wrox.chapter6.model.vo.ShowVO {
        public var title:String;
        public var description:String;
        public var availability:String;
}
```

www.it-ebooks.info

Using fscommands

Now you'll look at how `fscommands` is used.

FSCommandsProxy.as

The `FSCommandsProxy` is used to make calls to various `fscommand2()` commands that need to set settings in the application. In Listing 6-3 you'll see that the `FullScreen` and `SetFocusRectColor` commands are called when the proxy is registered.



LISTING 6-3: The FSCommandsProxy class for the TV Listings application

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter6.model.FSCommandsProxy extends Proxy implements IProxy {

      public static var NAME:String = "FSCommandsProxy";

      private var fscommand2:Function;

      public function FSCommandsProxy(data:Object) {
            super(NAME, data);
      }

      public function onRegister():Void {
            fscommand2("FullScreen", "true");
            fscommand2("SetFocusRectColor", "100", "179", "217");
      }
}
```

ResultsProxy.as

Unlike the `FSCommandsProxy`, the `ResultsProxy` does not need its `onRegister()` method implemented; instead, the proxy will be invoked when the user requests a search. The `ResultsProxy` will essentially need to serve two purposes:

- Let the application know it needs to make a request over the network.
- Let the application know when it has retrieved and parsed data.

Take note of the eight private variables declared for the `ResultsProxy` class, which is shown in Listing 6-4.

The `selectedIndex` variable is used to store a reference to the currently selected show, that is, the selected index in a results array. The variables `tvgSearchPhrase`, `tvgBroadcaster`, `tvgResultsPerPage`, and `tvgTrimDescription` were covered earlier, and with exception for `tvgSearchPhrase`, these are predefined. `tvgBroadcaster` is set to "OnlineTV", `tvgResultsPerPage` is set to "5", and `tvgTrimDescription` is set to "20". The `searchQuery` variable will be used to build a query string that will then be sent to the TV Genius server. The `url` variable is simply used to reference the path to the TV Genius server and is set to

www.it-ebooks.info

`http://listings.tvgenius.co.uk/feeds/search.xml?` Lastly, the `xml` variable is to reference the XML data requested from the TV Genius server.



LISTING 6-4: The ResultsProxy class

```
import com.wrox.chapter6.ApplicationFacade;
import com.wrox.chapter6.model.vo.ShowVO;
import mx.utils.Delegate;
import mx.xpath.XPathAPI;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter6.model.ResultsProxy extends Proxy implements IProxy {

      public static var NAME:String = "ResultsProxy";

      private var selectedIndex:Number;
      private var tvgSearchPhrase:String = "";
      private var tvgBroadcaster:String = "OnlineTV";
      private var tvgResultsPerPage:String = "5";
      private var tvgTrimDescription:String = "20";
      private var searchQuery:String = "";
      private var url:String = "http://listings.tvgenius.co.uk/feeds/search.xml?";
      private var xml:XML;

      public function ResultsProxy(data:Object) {
            super(NAME, data);
      }

      public function search(query:String):Void {
            searchQuery = url + "tvgSearchPhrase=" + query;
            searchQuery += "&tvgBroadcaster=" + tvgBroadcaster;
            searchQuery += "&tvgResultsPerPage=" + tvgResultsPerPage;
            searchQuery += "&tvgTrimDescription=" + tvgTrimDescription;

            xml = new XML();
            xml.ignoreWhite = true;
            xml.onLoad = Delegate.create(this, onLoadComplete);
            xml.load(searchQuery);
      }

      private function onLoadComplete(success:Boolean):Void {
            if(success){
                  parseXML(xml.firstChild);
            }
      }

      private function parseXML(xmlNode:XMLNode):Void {
            var dataArray:Array = [];
            var nodeStr:String = "/almostRss/channel/item";
            var xmlNodeList:Array = XPathAPI.selectNodeList(xmlNode, nodeStr);
```

continues

www.it-ebooks.info

LISTING 6-4 *(continued)*

```
            for(var i:Number=0; i < xmlNodeList.length; i++) {
                  dataArray.push(parseItemXMLNode(xmlNodeList[i]));
            }

            selectedIndex = 0;
            setData(dataArray);

            sendNotification(ApplicationFacade.DISPLAY_RESULTS, resultDetails());
      }

      private function parseItemXMLNode(node:XMLNode):ShowVO {
            var vo:ShowVO = new ShowVO();

            var childNode:XMLNode;
            childNode = XPathAPI.selectSingleNode(node, "item/title");
            vo.title = childNode.firstChild.nodeValue;
            childNode = XPathAPI.selectSingleNode(node, "item/description");
            vo.description = childNode.firstChild.nodeValue;
            childNode = XPathAPI.selectSingleNode(node, "item/availability");
            vo.availability = childNode.firstChild.nodeValue;

            return vo;
      }

      public function resultDetails():Array {

            var statusStr:String;
            statusStr = "Showing " + String(selectedIndex + 1) + " of ";
            statusStr += data.length + " results";

            var showStr:String;
            showStr = ShowVO(data[selectedIndex]).title + newline;
            showStr += ShowVO(data[selectedIndex]).availability;

            return [statusStr, showStr];
      }

      public function getSelected():ShowVO {
            return data[selectedIndex];
      }

      public function nextResult():Void {

            selectedIndex++;

            if (selectedIndex >= data.length) {
                  selectedIndex = 0;
            }
```

www.it-ebooks.info

```
            sendNotification(ApplicationFacade.SELECTED_RESULT_UPDATED, resultDetails());
        }

        public function previousResult():Void {

            selectedIndex--;

            if (selectedIndex <= -1) {
                selectedIndex = data.length--1;
            }

            sendNotification(ApplicationFacade.SELECTED_RESULT_UPDATED, resultDetails());
        }

        public function setResultAt(index:Number):Void {

            if (index >= data.length) {
                selectedIndex = 0;
            } else {
                selectedIndex = index;
            }

            sendNotification(ApplicationFacade.SELECTED_RESULT_UPDATED, resultDetails());
        }
    }
```

There is a lot going on inside this class, which needs to be explained in more detail in the next few sections. You'll return to the `ResultsProxy` shortly.

Using XML Data

There are a few chapters in the book that utilize XML as part of the model. In each instance you'll notice several core classes used, which you need to be aware of; for example, in `ResultsProxy`:

- `XML`
- `XPathAPI`
- `XMLNode`

Defining XML Objects

Parsing and using `XML` is a manual process which can require very specific references to obtain well-structured data. To do this, first you need to create an `XML` object and call its `load()` method, supplying a parameter to the XML document you want to load.

In order to utilize the data in an XML file, you need to parse it, and before you parse it, you need to load it. *How do you know when XML data has loaded?* You need to define a handler for the `onLoad` event, an event that is fired when the data has loaded. The event handler returns a `Boolean` value that indicates whether the data load was a *success* or *failure*, and on success you should be able to extract the data from the `XML` file and assign it to value objects for use.

www.it-ebooks.info

Here's the snippet of code that details the XML *load* for the TV Listings application:




```
xml = new XML();
xml.ignoreWhite = true;
xml.onLoad = Delegate.create(this, onXmlLoadComplete);
xml.load(searchQuery);
```

code snippet ResultsProxy.as

You'll notice here that there is a property called `ignoreWhite`. This property is set to `true`, which means that any white space found in the XML document will be ignored should it have any, so it is not treated as additional pieces of data.

Using XPathAPI

In AS2 there is a class you can import to help with parsing XML, called `XPathAPI`:

```
import mx.xpath.XPathAPI;
```

Using `XPathAPI` makes parsing XML is easy, by taking out a lot of the work required to reference data elements. `XPathAPI` allows you to filter XML results based on node names and attribute values in an XML file, and there are several static methods that can be used in various scenarios, including:

- `XPathAPI.selectNodeList()` — To return an array of `xmlNode` objects.
- `XPathAPI.selectSingleNode()` — To return a single `xmlNode` object.

These two methods are of particular interest, as these are used in the example TV Listings application.

The `XPathAPI.selectNodeList()` and `XPathAPI.selectSingleNode()` methods simply require two parameters. The first is an `xmlNode` object, which could be the `firstChild` of an XML document. This value indicates a starting point for `XPathAPI` to filter from. The second parameter is the `path`, which is supplied as a string representation of the path to the node value you wish to extract from the XML, using forward slash notation.

The following code snippet, taken from the TV listings example, shows how to implement `XPathAPI.selectNodeList()`:




```
var nodeStr:String = "/almostRss/channel/item";
var xmlNodeList:Array = XPathAPI.selectNodeList(xmlNode, nodeStr);
```

code snippet ResultsProxy.as

If you recall the XML snippet from earlier, you will be able to see how the node path determines the node values to place in the array. For instance, "/almostRss/channel/item" would correspond to both item nodes in the following snippet. In this example the `xmlNodeList.length()` would be equal to 2.

www.it-ebooks.info




```
<almostRss>
      <title></title>
      <channel>
            <item>
                  <title>Item 1</title>
                  <description>Description 1</description>
            </item>
            <item>
                  <title>Item 2</title>
                  <description>Description 2</description>
            </item>
      </channel>
</almostRss>
```

code snippet tvGenius.xml

Using XMLNode

While the `XPathAPI.selectNodeList()` returns an array containing any number of `xmlNode` objects that match the nodes of the `path` that you've supplied, `XPathAPI.selectSingleNode()`, as the name suggests, returns a single `xmlNode` based on the parameters supplied to the method.

The following snippet of code shows you how to implement `XPathAPI.selectSingleNode()`:




```
var childNode:XMLNode;
childNode = XPathAPI.selectSingleNode(node, "item/title");
vo.title = childNode.firstChild.nodeValue;
```

code snippet ResultsProxy.as

Following on from the previous code snippet, if the first found node from the `.xml` file `node` is supplied to the `XPathAPI.selectSingleNode()` method, along with the node path `item/title`, your result should set the value `Item 1` to the `vo.title` property. This should give you insight into how the XML will be parsed.




```
<item>
      <title>Item 1</title>
      <description>Description 1</description>
</item>
```

code snippet tvGenius.xml

ResultsProxy.as (a continuation)

Returning to the `ResultsProxy`, this class needs to step through each of the data nodes in the XML returned from the server. Several functions are used in the class. In the constructor for the proxy, where the XML object is created, you'll see that, when the `onLoad` event is triggered, it will call the `onLoadComplete()` function, passing a reference as to the whether the load was successful. If it was successful, a method called `parseXML()` is called (see Listing 6-5).

www.it-ebooks.info



LISTING 6-5: The onLoadComplete method for ResultsProxy

```
public function search(query:String):Void {
      searchQuery = url + "tvgSearchPhrase=" + query;
      searchQuery += "&tvgBroadcaster=" + tvgBroadcaster;
      searchQuery += "&tvgResultsPerPage=" + tvgResultsPerPage;
      searchQuery += "&tvgTrimDescription=" + tvgTrimDescription;

      xml = new XML();
      xml.ignoreWhite = true;
      xml.onLoad = Delegate.create(this, onLoadComplete);
      xml.load(searchQuery);
}

private function onLoadComplete(success:Boolean):Void {
      if(success){
            parseXML(xml.firstChild);
      }
}
```

The `parseXML()` is provided for parsing the `firstChild` property of an `.xml` file when it has successfully loaded (see Listing 6-6). In `parseXML()` you'll see that there is a `for` loop that pushes a value to a local `dataArray` array object. Here, a call to a function called `parseItemXMLNode()` is made, which returns an instance of `ShowVO`. The `dataArray` is populated with instances of `ShowVO` until there are no more nodes to parse, after which, the data is saved via `setData()`. Then, lastly, the `DISPLAY_RESULTS` notification is sent with an array in the notification body, returned from a method called `resultDetails()`. This will set the first result retrieved in for the `ResultsMediator`.



LISTING 6-6: The parseXML method for ResultsProxy

```
private function parseXML(xmlNode:XMLNode):Void {
      var dataArray:Array = [];
      var nodeStr:String = "/almostRss/channel/item";
      var xmlNodeList:Array = XPathAPI.selectNodeList(xmlNode, nodeStr);

      for(var i:Number=0; i < xmlNodeList.length; i++) {
            dataArray.push(parseItemXMLNode(xmlNodeList[i]));
      }

      selectedIndex = 0;
      setData(dataArray);

      sendNotification(ApplicationFacade.DISPLAY_RESULTS, resultDetails());
}
```

The `resultDetails()` method sets a status string property, `statusStr`, and a show string property, `showStr`, based on the `selectedIndex` variable, which is initially set to 0 in `parseXML()` (see Listing 6-7).

www.it-ebooks.info



LISTING 6-7: The resultDetails method for ResultsProxy

```
public function resultDetails():Array {

      var statusStr:String;
      statusStr = "Showing " + String(selectedIndex + 1) + " of ";
      statusStr += data.length + " results";

      var showStr:String;
      showStr = ShowVO(data[selectedIndex]).title + newline;
      showStr += ShowVO(data[selectedIndex]).availability;

      return [statusStr, showStr];
}
```

In `parseItemXMLNode()`, the `XPathAPI`, `XML`, and `XMLNode` classes are used to convert the raw `XML` result returned from the TV Genius server into manageable chunks of data. Here, the "item/title", "item/description", and "item/availability" node paths are retrieved and saved to their respective properties in the value object `ShowVO` (see Listing 6-8).



LISTING 6-8: The parseItemXMLNode method for ResultsProxy

```
private function parseItemXMLNode(node:XMLNode):ShowVO {
      var vo:ShowVO = new ShowVO();

      var childNode:XMLNode;
      childNode = XPathAPI.selectSingleNode(node, "item/title");
      vo.title = childNode.firstChild.nodeValue;
      childNode = XPathAPI.selectSingleNode(node, "item/description");
      vo.description = childNode.firstChild.nodeValue;
      childNode = XPathAPI.selectSingleNode(node, "item/availability");
      vo.availability = childNode.firstChild.nodeValue;

      return vo;
}
```

Within `ResultsProxy`, there are also four public methods defined to get the data:

- `getSelected()`
- `nextResult()`
- `previousResult()`
- `setResultAt()`

Some of these methods are similar to those created in Chapter 5.

The `getSelected()` method returns a specific instance of `ShowVO` from the `data` property of `ResultsProxy` (see Listing 6-9).

www.it-ebooks.info



LISTING 6-9: The getSelected method for ResultsProxy

```
public function getSelected():ShowVO {
      return data[selectedIndex];
}
```

In Listing 6-10 you'll see the functions for `nextResult()`, `previousResult()`, and `setResultAt()`. Each function sends the `SELECTED_RESULT_UPDATED` notification along with the details for the show via the `resultDetails()` method in the notification body. The public `nextResult()` and `previousResult()` functions are for handling calls to view the next and previous search results. The `nextResult()` call increases the value of the `selectedIndex` variable, while the `previousResult()` decreases the value of the `selectedIndex` variable. The `setResultAt()` method allows for a specific result to be referenced, setting the `selectedIndex` variable to the parameter passed to the method. Each method has a condition so that the `selectedIndex` value can't be invalid.



LISTING 6-10: The nextResult, previousResult and setResultAt methods for ResultsProxy

```
public function nextResult():Void {

      selectedIndex++;

      if (selectedIndex >= data.length) {
            selectedIndex = 0;
      }

      sendNotification(ApplicationFacade.SELECTED_RESULT_UPDATED, resultDetails());
}

public function previousResult():Void {

      selectedIndex--;

      if (selectedIndex <= -1) {
            selectedIndex = data.length--1;
      }

      sendNotification(ApplicationFacade.SELECTED_RESULT_UPDATED, resultDetails());
}

public function setResultAt(index:Number):Void {

      if (index >= data.length) {
            selectedIndex = 0;
      } else {
            selectedIndex = index;
      }

      sendNotification(ApplicationFacade.SELECTED_RESULT_UPDATED, resultDetails());
}
```

www.it-ebooks.info

This completes the `ResultsProxy`. Later you'll see how this proxy is utilized by the commands. Next, you'll cover the `NetworkProxy` class, which will be responsible for updating the user on Network Request and Connection statuses.

NetworkProxy.as

The `NetworkProxy` class consists of the `NetworkRequest`, `NetworkConnection`, and `Timer` classes, which were covered in Chapter 3. Both `NetworkRequest` and `NetworkConnection` classes have been updated so that, first, each time their `updateStatus()` methods are called the `fscommand2()` calls are made, then, second, values returned by their respective `fscommand2()` command calls can be saved and retrieved via two methods, called `saveMsg()` and `retrieveMsg()`. The `saveMsg()` replaces the `trace()` statements that were used previously, as shown for the `NetworkConnection` class in Listing 6-11.



LISTING 6-11: The NetworkConnection class for the TV Listings application

```
class com.wrox.chapter6.model.NetworkConnection {

      public static var FSCOMMAND2_NOT_SUPPORTED:Number = -1;
      public static var CONNECTED:Number = 0;
      public static var TRYING:Number = 1;
      public static var NOT_CONNECTED:Number = 2;
      public static var SUSPENDED:Number = 3;
      public static var INDETERMINABLE:Number = 4;

      private var _status:Number;
      private var _msg:String;

      public function NetworkConnection(){
           _status = fscommand2("GetNetworkConnectStatus");
      }

      public function updateStatus():Void {

            _status = fscommand2("GetNetworkConnectStatus");

            switch(_status){
                  case FSCOMMAND2_NOT_SUPPORTED:
                        fscommand2NotSupported();
                  break;
                  case CONNECTED:
                        saveMsg("Connected");
                  break;
                  case TRYING:
                        saveMsg("Trying To Connect");
                  break;
                  case NOT_CONNECTED:
                        saveMsg("Not Connected");
                  break;
                  case SUSPENDED:
                        saveMsg("Suspended");
                  break;
```

continues

www.it-ebooks.info

LISTING 6-11 *(continued)*

```
                case INDETERMINABLE:
                      saveMsg("Indeterminable");
                break;
          }
    }

    private function fscommand2NotSupported():Void {
          saveMsg("FSCOMMAND2 NOT SUPPORTED ON THIS DEVICE");
    }

    private function saveMsg(msg:String):Void {
          _msg = msg
    }

    public function retrieveMsg():String {
          return _msg;
    }
}
```

The `Timer` class has also been updated so that the tick count can be set, and an `onTick` event is dispatched when a second has elapsed (see Listing 6-12).



LISTING 6-12: The Timer class for the TV Listings application

```
import mx.utils.Delegate;
import mx.events.EventDispatcher;

class com.wrox.chapter6.model.Timer {

      public var addEventListener:Function;
      public var dispatchEvent:Function;
      public var removeEventListener:Function;

      private var duration:Number = 1000;
      private var tickCount:Number = 10;
      private var timerId:Number;

      public function Timer(timeInSeconds:Number){
            if(timeInSeconds != null) { tickCount = timeInSeconds; }
            EventDispatcher.initialize(this);
      }

      public function init():Void {
            trace("init():: tickCount = " + tickCount);
            timerId = setInterval(Delegate.create(this, tickComplete), duration);
      }

      public function tickComplete():Void {
            tickCount--;
```

www.it-ebooks.info

```
                trace("tickComplete():: tickCount = " + tickCount);
                if(tickCount == 0){
                      clearInterval(timerId);
                      dispatchEvent({target:this, type:"onTimerCompleted"});
                } else {
                      dispatchEvent({target:this, type:"onTick"});
                }
          }
    }
```

In Listing 6-13 you'll see the full `NetworkProxy` class. In the `onRegister()` method, instances of `NetworkConnection`, `NetworkRequest`, and `Timer` are created. The proxy has three functions. The `initTimer()` is where the proxy is initialized; here's where the event listeners are defined for the `timer` object. The timer is set to run for 30 seconds, during which the `sendUpdate()` function will be called in the proxy. When the timer counts down to 0, the `stopTimer()` method will be called, where the event listeners are removed from the `timer` object.



LISTING 6-13: The NetworkProxy class for the TV Listings application

```
import com.wrox.chapter6.ApplicationFacade;
import com.wrox.chapter6.model.NetworkConnection;
import com.wrox.chapter6.model.NetworkRequest;
import com.wrox.chapter6.model.Timer;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;
import mx.utils.Delegate;

class com.wrox.chapter6.model.NetworkProxy extends Proxy implements IProxy {

      public static var NAME:String = "NetworkProxy";

      private var connection:NetworkConnection;
      private var request:NetworkRequest;
      private var timer:Timer;
      private var update:String

      public function NetworkProxy(data:Object) {
            super(NAME, data);
      }

      public function onRegister():Void {
            connection = new NetworkConnection();
            request = new NetworkRequest();
      }

      public function initTimer():Void {
            timer = new Timer(30);
            timer.addEventListener("onTick", Delegate.create(this, sendUpdate));
            timer.addEventListener("onTimerCompleted", Delegate.create(this,
                  stopTimer));
```

continues

www.it-ebooks.info

LISTING 6-13 *(continued)*

```
            timer.init();
        }

        public function sendUpdate():Void {
            connection.updateStatus();
            request.updateStatus();
            update = "Connection Status: " + connection.retrieveMsg() + newline;
            update += "Request Status: " + request.retrieveMsg();
            trace("update");
            sendNotification(ApplicationFacade.NETWORK_STATUS_UPDATED, update);
        }

        public function stopTimer():Void {
            timer.removeEventListener("onTick", Delegate.create(this, sendUpdate));
            timer.removeEventListener("onTimerCompleted", Delegate.create(this,
                stopTimer));
        }
    }
```

In the `sendUpdate()` method, you'll see that the `NETWORK_STATUS_UPDATED` notification is sent with the connection and status updates in the `update` variable as part of the notification body. The aim here is to let the application know that network access is available.

You have now completed all the aspects of the model design highlighted earlier in Figure 6-1.

Creating the TV Listings Application's Controller

In this section you'll go through the commands used for the controller in the TV Listings application and cover:

- Preparing the application
- Displaying the view components
- Carrying out the search
- Navigating through the results
- Quitting the app

The design for the controller includes 10 commands, which as part of the *controller*, are collectively responsible for updating the application and responding to various notifications initiated by *events* driven by the user from the device.

Figure 6-3 shows the isolated concept of the controller.

www.it-ebooks.info

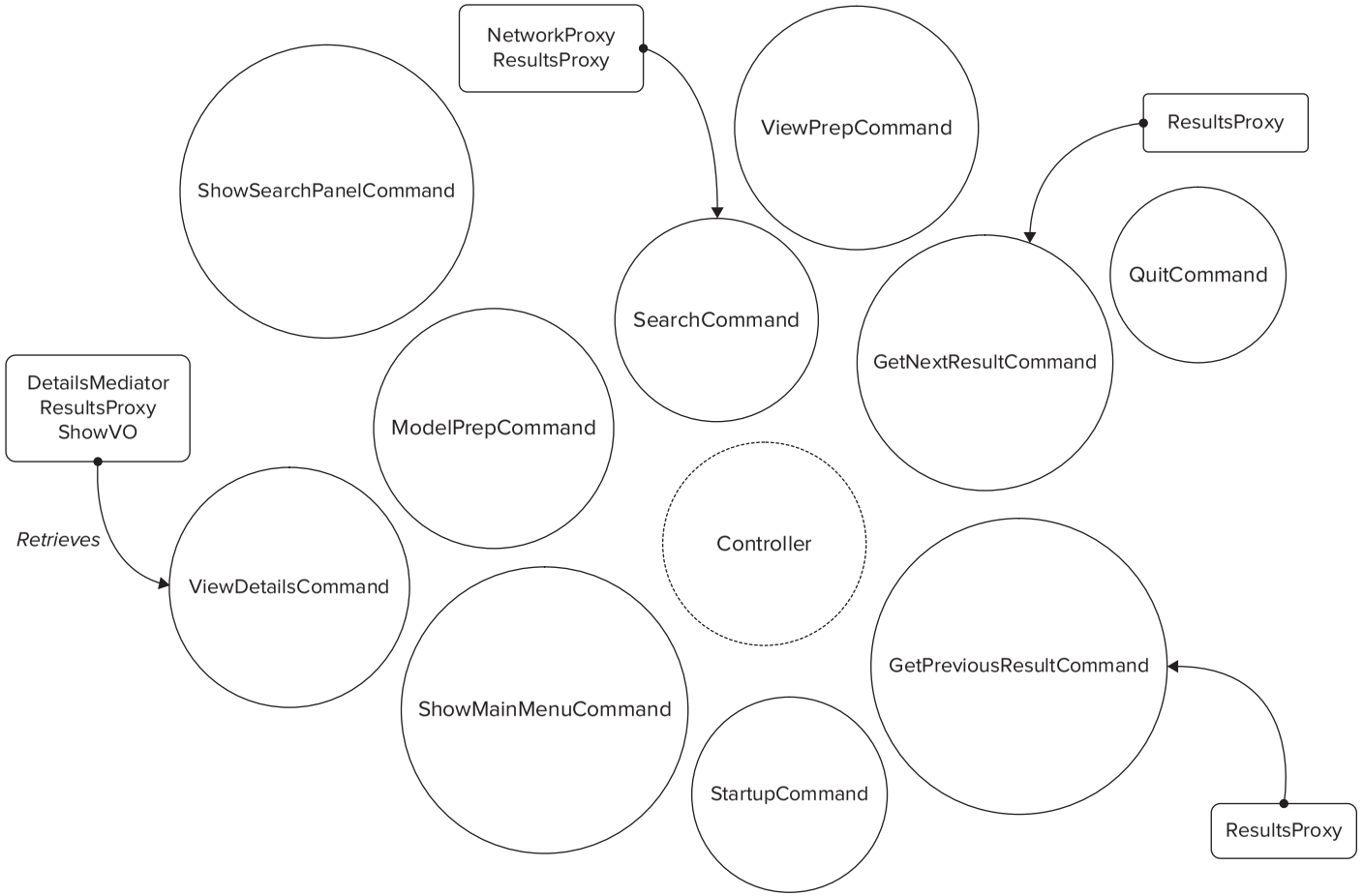


FIGURE 6-3

www.it-ebooks.info

The Project panel, shown in Figure 6-4, indicates the file system layout for the TV Listings application. Figure 6-4 shows each of the files under the controller directory.



FIGURE 6-4

Preparing the Application

The three commands responsible for the preparation of the application are `StartupCommand`, `ModelPrepCommand`, and `ViewPrepCOmmand`. The `StartupCommand` is responsible for getting your application ready. The command extends the `MacroCommand` class and initializes aspects of the model *via* `ModelPrepCommand` and the view *via* `ViewPrepCommand`.

StartupCommand.as

Listing 6-14 shows the full `StartupCommand` class. You'll notice that it is almost exactly the same as the `MacroCommand` used in Chapter 5, with the exception of the package name for the project's controller. Here, the sequence of commands to be executed via `addSubCommand()` are in the correct order, `ModelPrepCommand`, then `ViewPrepCommand`.

LISTING 6-14: The StartupCommand for the TV Listings application

Available for download on Wrox.com

```
import com.wrox.chapter6.controller.*;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.StartupCommand extends MacroCommand {
      private function initializeMacroCommand():Void {
            addSubCommand(new ModelPrepCommand());
            addSubCommand(new ViewPrepCommand());
      }
}
```

ModelPrepCommand.as

The `ModelPrepCommand` registers each of the concrete `Proxy` classes used for the application via the `facade.registerProxy()` method (see Listing 6-15).

LISTING 6-15: The ModelPrepCommand for the TV Listings application

Available for download on Wrox.com

```
import com.wrox.chapter6.model.*;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;
```

www.it-ebooks.info

```
class com.wrox.chapter6.controller.ModelPrepCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            facade.registerProxy(new FSCommandsProxy());
            facade.registerProxy(new ResultsProxy());
            facade.registerProxy(new NetworkProxy());
      }
}
```

ViewPrepCommand.as

After the `ModelPrepCommand` has readied the various proxies, the view can be initialized. The `ViewPrepCommand` class is responsible for registering mediators used in the application via the `facade.registerMediator()` call (see Listing 6-16). Each mediator will be covered later. The command also sends the `SHOW_MAIN_MENU` notification so that the `MainMenuMediator` is revealed when the application starts.



LISTING 6-16: The ViewPrepCommand for the TV Listings application

```
import com.wrox.chapter6.ApplicationFacade;
import com.wrox.chapter6.view.*;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.command.SimpleCommand;
import org.puremvc.as2.patterns.observer.Notification;

class com.wrox.chapter6.controller.ViewPrepCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            var mc:MovieClip = MovieClip(note.getBody());
            facade.registerMediator(new DetailsMediator(mc));
            facade.registerMediator(new MainMenuMediator(mc));
            facade.registerMediator(new NetworkMediator(mc));
            facade.registerMediator(new ResultsMediator(mc));
            facade.registerMediator(new SearchPanelMediator(mc));

            sendNotification(ApplicationFacade.SHOW_MAIN_MENU);
      }
}
```

Displaying the View Components

In this section we'll cover the commands that have a hand in revealing each of the view components from the library of the `.fla` file.

ShowSearchPanelCommand.as

The `ShowSearchPanelCommand` is mapped to the `SHOW_SEARCH_PANEL` notification. So when the user selects the "SEARCH" option in the menu, this command is invoked (see Listing 6-17).

www.it-ebooks.info



LISTING 6-17: The ShowSearchPanelCommand Class

```
import com.wrox.chapter6.view.SearchPanelMediator;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.ShowSearchPanelCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            var searchPanel:SearchPanelMediator;
            searchPanel = SearchPanelMediator(facade.retrieveMediator
(SearchPanelMediator.NAME));
            searchPanel.show();
      }
}
```

ShowMainMenuCommand.as

The `ShowMainMenuCommand` is mapped to the `SHOW_MAIN_MENU` notification. So when the user selects this option while the search is in view, this command is invoked and the user is presented with the main menu options (see Listing 6-18).



LISTING 6-18: The ShowMainMenuCommand class

```
import com.wrox.chapter6.view.MainMenuMediator;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.ShowMainMenuCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            var mainMenu:MainMenuMediator;
            mainMenu =
MainMenuMediator(facade.retrieveMediator(MainMenuMediator.NAME));
            mainMenu.show();
      }
}
```

ViewDetailsCommand.as

The `VIEW_DETAILS` notification, mapped to the `ViewDetailsCommand`, should allow the user to view the details of a selected TV Listings result. The command retrieves `DetailsMediator` and `ResultsProxy`, and then passes the `title` and `description` properties of the currently selected `ShowVO` instance as parameters for the `show()` method of the mediator (see Listing 6-19).



LISTING 6-19: The ViewDetailsCommand class

```
import com.wrox.chapter6.view.DetailsMediator;
import com.wrox.chapter6.model.ResultsProxy;
```

www.it-ebooks.info

```
import com.wrox.chapter6.model.vo.ShowVO;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.ViewDetailsCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            var details:DetailsMediator;
            details = DetailsMediator(facade.retrieveMediator(DetailsMediator.NAME));

            var rProxy:ResultsProxy;
            rProxy = ResultsProxy(facade.retrieveProxy(ResultsProxy.NAME));

            var title:String = ShowVO(rProxy.getSelected()).title;
            var description:String = rProxy.getSelected().description;
            details.show(title, description);
      }
}
```

Carrying Out the Search

Here you'll take a look at the command that is responsible for initiating the search.

SearchCommand.as

The `SearchCommand` retrieves the `NetworkProxy` and invokes the `initTimer()` method so that the details can be retrieved with regard to the status of the network request and connection for the `NetworkMediator`. The command initializes the search by retrieving the `ResultsProxy` and then invoking its `search()` function. In the `execute()` method for the command, you'll see that the notification body, consisting of a search query string, is supplied as a parameter for the method (see Listing 6-20). The query text used here is sent from a mediator called `SearchPanelMediator`, which will be covered shortly.



LISTING 6-20: The SearchCommand class

```
import com.wrox.chapter6.model.NetworkProxy;
import com.wrox.chapter6.model.ResultsProxy;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.observer.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.SearchCommand extends SimpleCommand {
      public function execute(note:INotification):Void {
            var queryStr:String = String(note.getBody());

            var nProxy:NetworkProxy;
            nProxy = NetworkProxy(facade.retrieveProxy(NetworkProxy.NAME));
            nProxy.initTimer();
```

continues

www.it-ebooks.info

LISTING 6-20 *(continued)*

```
            var rProxy:ResultsProxy;
            rProxy = ResultsProxy(facade.retrieveProxy(ResultsProxy.NAME));
            rProxy.search(queryStr);
      }
}
```

Navigating through the Results

Depending on how many results have been returned from the server, the user should be able to navigate through them one by one. The two commands `GetNextResultCommand` and `GetPreviousResultCommand`, mapped to the `GET_NEXT_RESULT` and `GET_PREV_RESULT`, respectively, are used for this purpose. At the end of the last chapter, you implemented a similar feature in the PureMVC example.

GetNextResultCommand.as and GetPreviousResultCommand.as

Both commands retrieve the `ResultsProxy`. `GetNextResultCommand` simply calls the `nextResult()` method on the `ResultsProxy` (see Listing 6-21).



LISTING 6-21: The GetNextResultCommand class

```
import com.wrox.chapter6.model.ResultsProxy;
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.GetNextResultCommand extends SimpleCommand
implements ICommand {
      public function execute(note:INotification):Void {
            var rProxy:ResultsProxy;
            rProxy = ResultsProxy(facade.retrieveProxy(ResultsProxy.NAME));
            rProxy.nextResult();
      }
}
```

The `GetPreviousResultCommand` calls the `previousResult()` method of `ResultsProxy` (see Listing 6-22).



LISTING 6-22: The execute method for GetPreviousResultCommand

```
public function execute(note:INotification):Void {
      var rProxy:ResultsProxy;
      rProxy = ResultsProxy(facade.retrieveProxy(ResultsProxy.NAME));
      rProxy.previousResult();
}
```

www.it-ebooks.info

Quitting the APP

Next take a look at the `QuitCommand`.

QuitCommand.as

As mentioned earlier, the `QuitCommand` is used to exit the application (see Listing 6-23).



LISTING 6-23: The QuitCommand class for the TV Listings application

```
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.command.*;

class com.wrox.chapter6.controller.QuitCommand extends SimpleCommand implements
ICommand {
      private var fscommand2:Function;

      public function execute(note:INotification):Void {
            fscommand2("Quit");
      }
}
```

Having covered each of the commands used in the application, you'll take a look at the view and its mediators next.

Exploring the View

The design for the view has five mediators; each presents information and responds to user input. In Figure 6-5 you'll see an isolated concept for the view used in the TV Listings application.

From Figure 6-5, you can see the mediators used in the TV Listings application:

- `MainMenuMediator` — To present options to the user at the start of the application.
- `SearchPanelMediator` — To present the search field required for user input.
- `ResultsMediator` — To present the results of the search based on the user's query.
- `DetailsMediator` — To present the details for an individual search result that are to be presented to the user separately.
- `NetworkMediator` – To present the network's status to the user.

www.it-ebooks.info

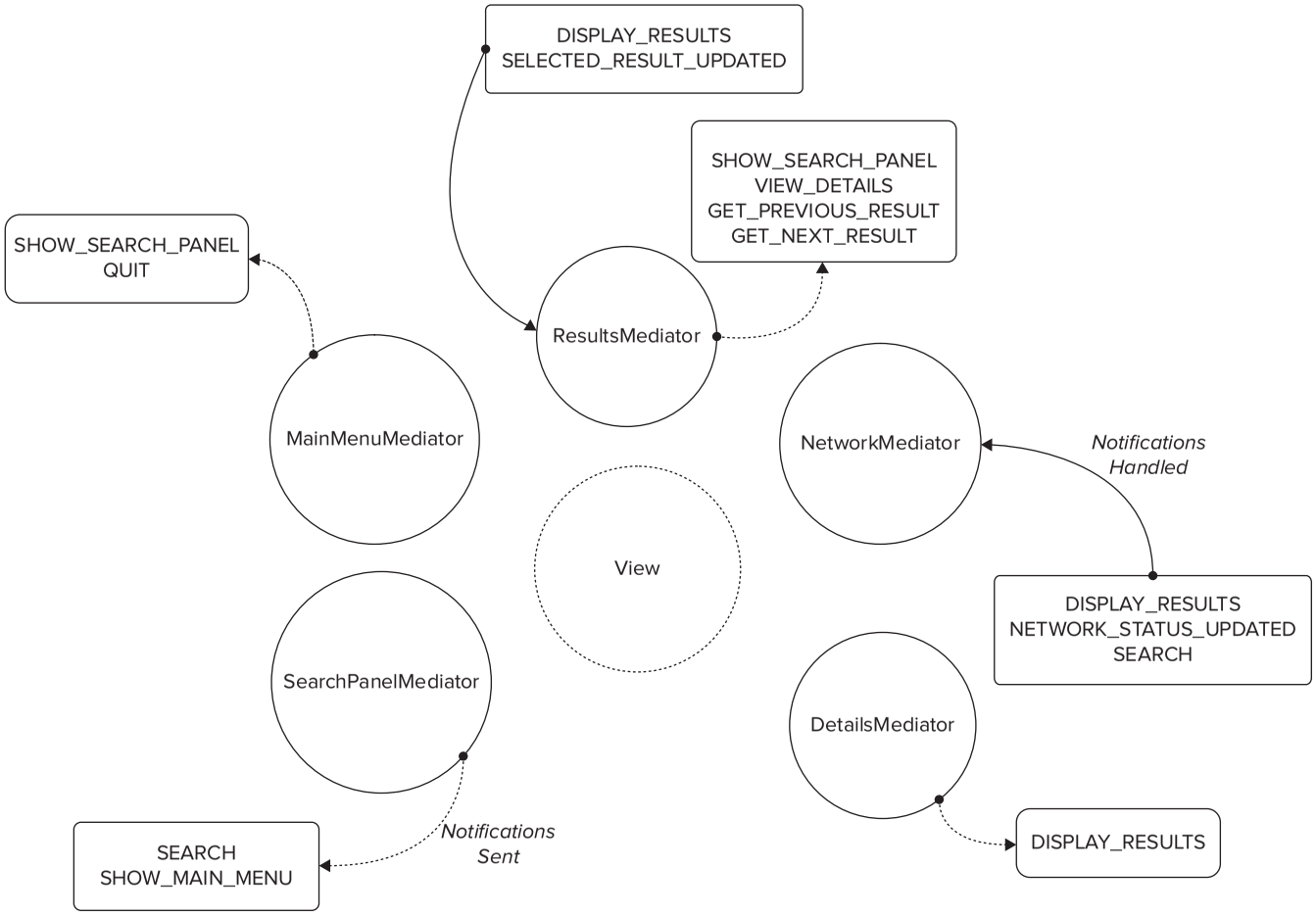


FIGURE 6-5

www.it-ebooks.info

The Project panel, shown in Figure 6-6, indicates the file system layout for the TV Listings application. Figure 6-6 shows each of the files under the view directory.



FIGURE 6-6

Displaying View Elements

As you go through each of the view elements in this section, you'll notice that they have a similar structure. Each of the components used is attached from the library of the `.fla` file. Each has its own implementation of a `show()` and `hide()` method, which among other things handles the positioning for the mediator. Lastly, each component from the library has a static text field for the title of the view, and for the labels representing the soft key presses, where appropriate.

MainMenuMediator.as

The `MainMenuMediator` is the first mediator to be revealed in the application when the `SHOW_MAIN_MENU` notification is sent. The mediator attaches the `menuContainer` MovieClip asset from the library of the `.fla` file for the app. The mediator contains two buttons, called `menuBtn1` and `menuBtn2`. As previously mentioned, the `ShowMainMenuCommand` retrieves the mediator, calling its `show()` method, which subsequently is where the mediator assigns methods for the `onRelease()` event handlers for the two items in the menu. The first button in the menu calls the `search()` method of the mediator, which sends `SHOW_SEARCH_PANEL` notification, while the second button calls the `quit()` method, sending the `QUIT` notification (see Listing 6-24).



Available for download on Wrox.com

LISTING 6-24: The MainMenuMediator class

```
import com.wrox.chapter6.ApplicationFacade;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;
import mx.utils.Delegate;

class com.wrox.chapter6.view.MainMenuMediator extends Mediator implements IMediator
{

      public static var NAME:String = "MenuMainMediator";

      private var menu:MovieClip;

      public function MainMenuMediator(viewComponent:Object) {
            super(NAME, viewComponent);
      }

      public function onRegister():Void {
            var depth:Number = _component.getNextHighestDepth();
            menu = _component.attachMovie("menuContainer", "menu", depth);
            hide();
```

continues

www.it-ebooks.info

LISTING 6-24 *(continued)*

```
        }

        public function show():Void {
              menu._x = 20;
              menu._y = 60;
              menu.items.menuBtn1.onRelease = Delegate.create(this, search);
              menu.items.menuBtn2.onRelease = Delegate.create(this, quit);
        }

        public function hide():Void {
              menu._x = Stage.width;
              menu._y = Stage.height;
        }

        public function search():Void {
              hide();
              sendNotification(ApplicationFacade.SHOW_SEARCH_PANEL);
        }

        public function quit():Void {
              sendNotification(ApplicationFacade.QUIT);
        }

        public function get _component():MovieClip {
              return MovieClip(viewComponent);
        }
}
```

SearchPanelMediator.as

In Listing 6-25 you'll see the code for `SearchPanelMediator`, which simply displays a dynamic text box and a button for the user to submit a query. The mediator implements the `Key` event listener's `onKeyDown()` method. In `onKeyDown()` the `ExtendedKey.SOFT1` is captured to hide the mediator, by calling `hide()`, and send the `SHOW_MAIN_MENU` notification. The `show()` method is called when the mediator is retrieved in the `ShowSearchPanelCommand`. The main role of the `SearchPanelMediator` is to capture and send the user's search query. The mediator attaches the `searchContainer` MovieClip from the library of the `.fla`, which contains the text field `searchTxt`. A button called `submitBtn`, also in the `searchContainer`, has the `onSubmit()` method delegated to the `onRelease()` event handler. In `onSumbit()`, the `SEARCH` notification is sent along with `searchTxt.text` property in the notification body. There are no notifications handled in the mediator.



LISTING 6-25: The SearchPanelMediator for the TV Listings application

```
import com.wrox.chapter6.ApplicationFacade;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;
import mx.utils.Delegate;
```

www.it-ebooks.info

```
class com.wrox.chapter6.view.SearchPanelMediator extends Mediator implements
IMediator {

      public static var NAME:String = "SearchPanelMediator";

      private var search:MovieClip;

      public function SearchPanelMediator(viewComponent:Object) {
            super(NAME, viewComponent);
      }

      public function onRegister():Void {
            var depth:Number = _component.getNextHighestDepth();
            search = _component.attachMovie("searchContainer", "search", depth);
            hide();
      }

      public function onSubmit():Void {
            hide();
            sendNotification(ApplicationFacade.SEARCH, search.searchTxt.text);
      }

      public function show():Void {
            search._x = 15;
            search._y = 60;
            search.searchBtn.onRelease = Delegate.create(this, onSubmit);
            Key.addListener(this);
      }

      public function hide():Void {
            search._x = Stage.width;
            search._y = Stage.height;
            Key.removeListener(this);
      }

      public function onKeyDown():Void {
            switch(Key.getCode()) {
                  case ExtendedKey.SOFT1:
                        hide();
                        sendNotification(ApplicationFacade.SHOW_MAIN_MENU);
                  break;
            }
      }

      public function get _component():MovieClip {
            return MovieClip(viewComponent);
      }
}
```

If you remember, the `SEARCH` notification invokes the `SearchCommand`, which is responsible for initializing searches via the `ResultsProxy`.

www.it-ebooks.info

ResultsMediator.as

The `ResultsMediator` will display the results from a user's search query and has several purposes: to display the results it retrieves from the server, displaying one result at a time; to show the number of results that the user can scroll through; to provide the ability to view the next and previous result; to allow the user to view the description of the result; and lastly to return to the search panel. The `ResultsMediator`, therefore, needs to be responsible for much of the user interaction, hence it implements a `Key` listener object (see Listing 6-26).

The mediator consists of the `resultsContainer` asset in the `.fla` file, which has two text fields, one to display the overall status and one to display information about the show.

The mediator implements the `Key` listener `onKeyDown()` method, which is added when the mediator is shown via `show()` and then removed via `hide()`. In `onKeyDown()` there are several keys captured. First, the `ExtendedKey.SOFT1` is captured to hide the mediator and send the `SHOW_SEARCH_PANEL` notification. The `ExtendedKey.SOFT2` and `Key.ENTER` keys are also captured to hide the mediator. They also send the `VIEW_DETAILS` notification. The `Key.UP` and `Key.LEFT` keys are captured to send the `GET_PREVIOUS_RESULT` notification, and lastly the `Key.DOWN` and `Key.RIGHT` keys are captured to send the `GET_NEXT_RESULT` notification.

You'll see that the mediator is interested in two notifications: `DISPLAY_RESULTS` and `SELECTED_RESULT_UPDATED`. When the `DISPLAY_RESULTS` notification is sent from the `ResultsProxy`, the status and the show information are retrieved from the notification body and set in the text fields. When the `SELECTED_RESULT_UPDATED` notification is sent from the `ResultsProxy`, the status and show information are updated.


Available for download on Wrox.com

LISTING 6-26: The ResultsMediator for the TV Listings application

```
import com.wrox.chapter6.ApplicationFacade;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter6.view.ResultsMediator extends Mediator implements IMediator {

      public static var NAME:String = "ResultsMediator";
      private var results:MovieClip;

      public function ResultsMediator(viewComponent:Object){
            super(NAME, viewComponent);
      }

      public function onRegister():Void {
            var depth:Number = _component.getNextHighestDepth();
            results = _component.attachMovie("resultsContainer", "results", depth);
            hide();
      }

      public function listNotificationInterests():Array {
```

www.it-ebooks.info

```
        return [ApplicationFacade.DISPLAY_RESULTS,
                ApplicationFacade.SELECTED_RESULT_UPDATED];
    }

    public function handleNotification(note:INotification):Void {
        switch(note.getName()){
            case ApplicationFacade.DISPLAY_RESULTS:
                if(note.getBody() != null){
                    results.statusTxt.text = note.getBody()[0];
                    results.showTxt.text = note.getBody()[1];
                }
                show();
            break;
            case ApplicationFacade.SELECTED_RESULT_UPDATED:
                if(note.getBody() != null){
                    results.statusTxt.text = note.getBody()[0];
                    results.showTxt.text = note.getBody()[1];
                }
            break;
        }
    }

    public function show():Void {
        results._x = 15;
        results._y = 60;
        Key.addListener(this);
    }

    public function hide():Void {
        results._x = Stage.width;
        results._y = Stage.height;
        Key.removeListener(this);
    }

    public function onKeyDown():Void {
        switch(Key.getCode()) {
            case ExtendedKey.SOFT1:
                hide();
                sendNotification(ApplicationFacade.SHOW_SEARCH_PANEL);
            break;
            case ExtendedKey.SOFT2:
            case Key.ENTER:
                hide();
                sendNotification(ApplicationFacade.VIEW_DETAILS);
            break;
            case Key.UP:
            case Key.LEFT:
                sendNotification(ApplicationFacade.GET_PREVIOUS_RESULT);
            break;
            case Key.DOWN:
            case Key.RIGHT:
                sendNotification(ApplicationFacade.GET_NEXT_RESULT);
```

continues

www.it-ebooks.info

LISTING 6-26 *(continued)*

```
                    break;
              }
        }

        public function get _component():MovieClip {
              return MovieClip(viewComponent);
        }
  }
```

When the user clicks on a selected item, the `VIEW_DETAILS` notification is sent and the `DetailsMediator` is invoked.

DetailsMediator.as

The `DetailsMediator`, shown in Listing 6-27, is responsible for providing the description of each result.

The mediator also implements the `Key` listener `onKeyDown()` method, added when the mediator is shown via `show()` and removed when the mediator is hidden via `hide()`. In `onKeyDown()` the `ExtendedKey.SOFT1` is captured, where the `DISPLAY_RESULTS` notification is sent.

If you remember, this mediator is retrieved in `ViewDetailsCommand`, so there are no notifications handled. Instead, the details presented to the user are supplied as parameters to the `show()` method, where the `title` and `description` parameters are utilized.



LISTING 6-27: The DetailsMediator class for the TV Listings application

```
import com.wrox.chapter6.ApplicationFacade;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter6.view.DetailsMediator extends Mediator implements IMediator {

      public static var NAME:String = "DetailsMediator";

      private var details:MovieClip;

      public function DetailsMediator(viewComponent:Object) {
            super(NAME, viewComponent);
      }

      public function onRegister():Void {
            var depth:Number = _component.getNextHighestDepth();
            details = _component.attachMovie("detailsContainer", "details", depth);
            hide();
      }

      public function show(title:String, description:String):Void {
            details._x = 15;
```

www.it-ebooks.info

```
            details._y = 60;
            details.titleTxt.text = title;
            details.descriptionTxt.text = description;
            Key.addListener(this);
      }

      public function hide():Void {
            details._x = Stage.width;
            details._y = Stage.height;
            Key.removeListener(this);
      }

      public function onKeyDown():Void {
            switch(Key.getCode()){
                  case ExtendedKey.SOFT1:
                        hide();
                        sendNotification(ApplicationFacade.DISPLAY_RESULTS);
                  break;
            }
      }

      public function get _component():MovieClip {
            return MovieClip(viewComponent);
      }
}
```

NetworkMediator.as

The `NetworkMediator`, shown in Listing 6-28, consists of the `networkContainer` asset, which is attached to the app from the library of the `.fla` file when the mediator is registered. The mediator lists an interest in three notifications: `DISPLAY_RESULTS`, `NETWORK_STATUS_UPDATED`, and `SEARCH`. The main role of the mediator is to display updates to the network status sent from the `NetworkProxy`, which it retrieves in the `NETWORK_STATUS_UPDATED` notification body. The `SEARCH` notification invokes the `show()` method, while the `DISPLAY_RESULTS` invokes its `hide()` method.



LISTING 6-28: The NetworkMediator for the TV Listings application

```
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;
import com.wrox.chapter6.ApplicationFacade;

class com.wrox.chapter6.view.NetworkMediator extends Mediator implements IMediator {

      public static var NAME:String = "NetworkMediator";

      private var networkStatus:MovieClip;

      public function NetworkMediator(viewComponent:Object) {
            super(NAME, viewComponent);
      }

      public function onRegister():Void {
```

continues

www.it-ebooks.info

LISTING 6-28 *(continued)*

```
            networkStatus = _component.attachMovie("networkContainer",
            "networkContainer", _component.getNextHighestDepth());
            hide();
      }

      public function show():Void {
            networkStatus._x = 20;
            networkStatus._y = 60;
      }

      public function hide():Void {
            networkStatus._x = Stage.width;
            networkStatus._y = Stage.height;
      }

      public function listNotificationInterests():Array {
            return [ApplicationFacade.DISPLAY_RESULTS,
                    ApplicationFacade.NETWORK_STATUS_UPDATED,
                    ApplicationFacade.SEARCH];
      }

      public function handleNotification(note:INotification):Void {
            switch(note.getName()){
                  case ApplicationFacade.DISPLAY_RESULTS:
                        hide();
                  break;
                  case ApplicationFacade.NETWORK_STATUS_UPDATED:
                        var statusStr:String = String(note.getBody());
                        networkStatus.statusTxt.text = statusStr;
                  break;
                  case ApplicationFacade.SEARCH:
                        show();
                  break;
            }
      }

      public function get _component():MovieClip {
            return MovieClip(viewComponent);
      }
}
```

There you have it, all PureMVC classes accounted for.

Viewing the Application in Device Central

You have now covered each aspect of the TV Listings application. Run the completed app in Device Central, and ensure that you have selected the Nokia E71 for testing.

www.it-ebooks.info

Figure 6-7 shows the resulting search for "Star Wars" in the results view for the TV Listings application on the Nokia E71 in Device Central.

The search for shows on "Star Wars" returns five results, which is the value set to the `tvgResultsPerPage` parameter. As an extended feature for the TV Listings application, this parameter and others could be changed so that their values could be dynamically set by the user.

Figure 6-8 shows the detail view for the "Star Wars: The Clone Wars" search, which simply shows the title and the show's description.



FIGURE 6-7



FIGURE 6-8

Again, as an additional feature, the detail view in the application could be modified with minimal changes so that the `DetailsMediator` registered an interest in the `GET_PREVIOUS_RESULT` and `GET_NEXT_RESULT` methods, allowing the user to navigate through the descriptions of the results without having to navigate back to the results view.

In the next section you'll take a look at some of the security issues with loading external data for Flash mobile applications.

SECURITY CONSIDERATIONS: LOADING DATA

While developing or testing your mobile applications, you may run into a situation where you're calling data and nothing is being returned, even though you are certain you are hitting the right URL, which can be very frustrating.

www.it-ebooks.info

Here, you'll take a look at `crossdomain.xml` and how to change the Playback security and Sandbox settings for the Flash Lite player, a few areas that can lead to pitfalls when attempting to load data.

The crossdomain.xml File

The `crossdomain.xml` file is a "policy file," an XML document that details whether a Flash `.swf` file can have access to data on a particular domain.

Here's an example of a policy file in use:

```
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-
domain-policy.dtd">
<cross-domain-policy>
      <site-control permitted-cross-domain-policies="all"/>
      <allow-access-from domain="*.tvgenius.net" />
</cross-domain-policy>
```

code snippet crossdomain.xml

Whenever a data call is made to a remote server or another domain, the Flash Lite 3.*x* player looks for this `crossdomain.xml` file and attempts to load it. If the policy file has the domain of the document within it, then data is accessible from the Flash movie calling the data. If not, then the request fails silently. The data can be anything. The following data calls will require a policy file to be in place on a domain if you make a request using Flash Lite 3.1:

- `loadVars()`
- `loadMovie()`
- `XML.load()`

Setting the Sandbox Type

In the Security panel of the Device Central Emulator, there are two options that allow you to set the Sandbox type for the application while you are testing. When "Local Trusted Sandbox" is selected, you can pretty much load data from any domain, while setting the Security option to "Default Sandbox" will remove the local trust settings and employ default settings for any application running in that environment. For instance, if there is a policy file on a server that only permits access to its data from a particular domain, and your `.swf` file isn't on that domain, you cannot retrieve data from that server.

Setting the Playback Security

The Flash Lite 3.1 player has its own security setting, which allows it to authenticate whether it can access particular content locally or remotely. This can be set via the File ➪ Publish Settings ➪ Flash tab. Near the bottom of the panel, the Local playback security field gives you two options to

www.it-ebooks.info

determine whether your application can use data remotely, that is, on the device or remotely via a server. Figure 6-9 shows the Publish Settings panel, where this can be edited.

Select "Access local files only" (those in the same domain) or "Access network only" to connect to a remote server. You need to ensure that this field is set correctly before testing your application on your device. Otherwise, remote access to TV Genius will be denied by the Flash Lite Player.



FIGURE 6-9

SUMMARY

In this chapter, you created an application that uses the TV Genius search engine to return TV listings based on the search terms you provide. You also learned how to parse XML data and utilize the `XPathAPI` and `XMLNode` classes to provide data in your Flash Lite applications. You also learned about the security issues.

In the next chapter, you take a look at creating a Flash Lite Media console.

www.it-ebooks.info

7

Creating a Media Console

WHAT'S IN THIS CHAPTER?

- Working with video and sound
- Utilizing NetConnection and NetStream classes
- Using Flash Media Server to stream media files
- Using PureMVC to build the Media Console

Popular music download services such as Sky Songs, Spotify, and iTunes lead the way in streaming media technology, providing services to millions of users every day, through a variety of interfaces including Web, desktop, and mobile. Audio and video continues to play a key role in today's rich media, content-driven Internet. In this chapter you'll learn how to utilize the media capabilities of Flash Lite and explore the build of a Flash Lite Media Console mobile application.

You will learn about both video and sound capabilities of the Flash Lite player, covering playback of content in Device Central.

THE CORE MEDIA CLASSES

First, let's detail the classes responsible for the playback of media in the Flash Lite 3.*x* Player, before moving on to creating an application. Central to the development of a media component is the `NetConnection` and `NetStream` classes. These must be used together in order to successfully play back content.

Using NetConnection

The `NetConnection` class can be used for connecting to your media, whether locally on the device or on a remote server. The class has two methods:

- `connect()` — Opens a connection
- `close()` — Closes a connection

www.it-ebooks.info

Here's an example of the net connection in use:

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
nc.close();
```

In this example you can see that `null` is supplied as the parameter to the connect method, which indicates that it is not connecting to a server. With a server, you need to pass the location of the server. In order to play a media file, you must first create an instance of the `NetConnection`, then open a connection to a location, and then provide the `NetConnection` object to the constructor of the `NetStream` on instantiation.

Using NetStream

The `NetStream` class is directly responsible for delivering media streams to the Flash Player and has several methods, properties, and events associated with its use. The class has numerous attributes, which can be grouped into several areas of responsibility, including:

- Handling playback
- Buffer monitoring
- Retrieving meta data
- Data stream loading

Handling Playback

The methods, properties, and event handlers associated with playback are listed here:

- `play()` — Playing the stream
- `pause()` — Pausing the stream
- `seek()` — Jump to a position in the stream
- `close()` — Close the stream
- `onStatus` — An event handler, which is invoked when the status of the `NetStream` is updated

To initiate playback, you need to create an instance of the `NetStream` object and supply a `NetConnection` instance as a parameter when it is instantiated. Here's an example demonstrating video playback of an `.flv` file:




```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.play("media/modernWarfare2.flv");
```

code snippet chapter7_videoPlayback.fla

www.it-ebooks.info

Buffer Monitoring

The buffer is the period of time and amount of stream data stored by your application. During the playback of a stream, you can retrieve the `bufferLength` and `bufferTime` properties of the `NetStream` object and use these values to determine or monitor the amount of buffer available. The buffer time is measured in seconds and the value assigned for the default is one tenth of a second or 0.1.

The three main attributes of the `NetStream` class that are responsible for buffer are:

- `setBufferTime()` — A method that allows you to set how long to buffer data before playing the stream. It takes a single parameter, which represents the number of seconds.
- `bufferLength` — A read-only property that returns the number of seconds of data currently in the buffer.
- `bufferTime` — A read-only property that returns the time assigned to the buffer, in seconds, when you set a value `setBufferTime()` method.

Determining Playback Progress

Another important aspect of the `NetStream` object is the `time` property. This can be used to determine the exact position of the media during playback. Because this property returns the time in seconds, to represent the progress in "mm:ss" format, you need to convert the seconds using the static `Math.floor()` method.

In the sample code that follows, you can see an example of the minutes and seconds being traced from the `NetStream` object:




```
import mx.utils.Delegate;

var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.play("media/modernWarfare2.flv");

var interval:Number = setInterval(Delegate.create(this, determineTime), 500, ns);

function determineTime(ns:NetStream) {
      var minutes:String = Math.floor(ns.time/60).toString();
      var seconds:String = Math.floor(ns.time%60).toString();

      if (seconds < 10) seconds = "0" + seconds;
      if (minutes < 10) minutes = "0" + minutes;

      trace( minutes + ":" + seconds );
}
```

code snippet chapter7_netStreamProgress.fla

www.it-ebooks.info

Here, you retrieve the time property from the `NetStream` object. The example utilizes `setInterval()`, where the `NetStream` instance is added as a third parameter to the function. This ensures that the `NetStream` object is always referenced in the `determineTime()` method, where subsequently the `time` property can be accessed.

Determining the Data Load

Another important aspect of the `NetStream` object is the information it can provide in detailing how much data has been loaded via the net stream connection. You don't want users waiting forever for audio or video to start just because they are not aware of how much data has been loaded. If you create a progress bar, you can aid users by keeping them informed of data loaded.

There are two properties of the `NetStream` class that can help in this scenario:

- `bytesLoaded` — Returns the amount of data loaded into the Flash Lite player via the `NetStream` object. Measured in KB.
- `bytesTotal` — Returns the total file size of the data stream. Measured in KB.

Retrieving Content Meta Data

You can use `onMetaData`, an event handler, which is invoked when the meta data of the `NetStream` is retrieved for the `NetStream` class to retrieve valuable information about a file being streamed. The `onMetaData()` method is invoked when the `NetStream` object's `play()` method is initiated. In order to use the data returned by `onMetaData()`, you must assign a `function` or delegate a method as the `onMetaData()` handler.

Here is sample code that can be used to retrieve each of the properties mentioned using `onMetaData`:




```
import mx.utils.Delegate;

var nc = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.onMetaData = Delegate.create(this, onMetaDataHandler);

function onMetaDataHandler(metaInfo:Object) {
      for (var propertyName:String in metaInfo) {
            trace(propertyName + " = " + metaInfo[propertyName]);
      }
}

ns.play("media/modernWarfare2.flv");
```

code snippet chapter7_metaData.fla

The following snippet shows the meta data that is traced in the output window when the example in `chapter7_videoMetaData.fla` is run in Device Central:

www.it-ebooks.info




```
audiosize = 1028597
audiodatarate = 124.267993420035
audiocodecid = 2
videosize = 4713827
framerate = 60
videodatarate = 569.492058229982
height = 405
width = 720
videocodecid = 4
canSeekToEnd = 1
metadatacreator = FlixEngineLinux_8.0.14.1 (www.on2.com)
datasize = 5742424
lasttimestamp = 64.666
duration = 64.666
```

code snippet metaData.txt

The information returned by the `metaInfo` object includes:

- `width` — The number of pixels representing the actual width of the video stream encoded
- `height` — The number of pixels representing the actual height of the video stream encoded
- `audioDataRate` — The number representing the quality of the audio, measured in kilobits per second (Kbits/s)
- `videoDataRate` — The number representing the quality of the video, measured in Kbits/second
- `frameRate` — The value representing the number of frames encoded and captured from the source video.
- `audioCodecId` — The name of the audio encoder
- `videoCodecId` — The name of the video encoder
- `duration` — The total length of the media
- `canSeekToEnd` — This Boolean value indicates whether you can progress through the media file to the end.
- `creationDate` — The date of the when the media was created

In the sample code you can see that the event handler takes a `metaInfo` object as a parameter. The `for` loop then iterates through the properties of `metaInfo` to obtain each value. The `width` and `height` properties of the `metaInfo` object, along with `frameRate`, `videoDataRate` and `videoCodecId`, and `audioCodeId` each refer to video-encoding properties. These are primarily for the encoding processes but could aid in determining whether the quality of playback could be affected by anything in particular.

For instance, the `width` and `height` property of an encoded video could quite easily vary based on the source file, and these properties can determine the aspect ratio of the playback. Hence, you can adjust the aspect ratio of the video by using the properties of the width and height values set in the meta data.

www.it-ebooks.info

This brings us nicely to the native `Video` class. In order to play back a `NetStream` object, you must use the `Video` object.

Using the Video Object

In order to play back and view video content in your applications, you need to use an instance of the `Video` object, which can be created either on stage or as a symbol in the library.

Creating a Video Object

To create the `Video` object:

1. In Flash CS4, right-click on the Library panel to open the Library options menu, then select New Video.
2. In the Video Properties window that opens, create a symbol name for the `Video` object, for example, "video." Ensure that you select as the "type" ActionScript controlled. This will allow you to manipulate the `Video` object at runtime. The alternative option allows you to "Embed the Video" directly into your application.

   Selecting the ActionScript controlled type also allows you to bundle the video source into a SWF file for use, but this can result in memory overload.
3. Once you click OK, you have a reference to the video instance in your library.
4. In order to use the instance dynamically, you need to create a container with which to attach the instance to the Stage at runtime. For this you use the `MovieClip`.
5. Create a `MovieClip` symbol, and drag the `Video` object instance inside the new symbol. Provide an identifier for the `MovieClip` object in the linkage, and ensure that Export for ActionScript and Export in frame 1 are selected.



You need to give a linkage name to the `Video` *object. In the example, double-click the "video" symbol in the library, then click on the* `Video` *object on the stage. Open the properties for the symbol, and if you see* `<Instance Name>`*, you know you haven't linked the video to the ActionScript. Hence, you may notice that you can hear audio but not see the video. This brings us to one of the features of audio with the Flash Lite player. With audio you are able to play through a* `MovieClip` *instance. An Audio object doesn't need to be created in the same way as video. In fact, all you need are the* `NetStream` *and* `NetConnection`*.*

Combining all the previous information on `NetConnection` and `NetStream`, you should now be able to play back video. The following example shows you how to use a `Video` object called `videoObj` from the library:

www.it-ebooks.info




```
var depth:Number = this.getNextHighestDepth()
var videoContainer:MovieClip = this.attachMovie("videoObj", "videoObj", depth);

var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);

var video:Video = videoContainer.video;
video.attachVideo(ns);

ns.play("media/modernWarfare2.flv");
```

code snippet chapter7_videoPlayback.fla

Later you will learn how to control video using ActionScript. Now let's turn our attention to audio and the `Sound` class.

Using the Sound Object

Here, you'll learn how to play back audio files for your applications. Essentially, the `Sound` class allows you to manage audio in the Flash player. While you can create sound instances to play directly on the timeline, or within a single frame, here you'll concentrate on using AS.

Unlike the `Video` class, you don't need to have a reference to the `Sound` object in the Library panel. The `Sound` object simply has three properties: `duration`, `position`, and `id3`. While you can reference an `id` of a sound clip from the library, you can instantiate the `Sound` class and load external audio files directly within a `MovieClip` instance using AS. There are various methods associated with using the `Sound` object that you should be aware of, which you'll cover shortly:

- `attachSound()`
- `loadSound()`
- `start()`
- `stop()`

Playing a Sound File from the Library

You can play a sound file by including it in your application. In order to do this, you must import the sound file into the library and assign a linkage identifier. The linkage ID can then be associated with the `Sound` object through the `attachSound()` method.




```
var sound:Sound = new Sound();
sound.attachSound("sound_linkage_ID");
sound.start();
```

code snippet chapter7_soundLinkage.fla

The sound file in the library with the linkage ID `sound_linkage_ID` will play instantly.

www.it-ebooks.info

Loading an External .mp3 File

Because of device memory limitations with the Flash Lite player, embed sound files in your applications with caution. Playing an external file is more suitable and the next listing shows you exactly how you to do just that (see Listing 7-1).



LISTING 7-1: Using the sound object to play an external .mp3 file

```
import mx.utils.Delegate;

var sound:Sound = new Sound();
sound.onLoad = Delegate.create(this, onLoadHandler);
sound.loadSound("media/track1.mp3", false);

function onLoadHandler(hasLoaded:Boolean){
      if(hasLoaded){
            trace("Sound file has loaded!");
            trace("Start it from the beginning!");
            sound.start(0, 1);
      } else {
            trace("Sound file has not loaded!");
      }
}
```

In this example you can see the sound object requires you to load the `track1.mp3` file via the `loadSound()` method, rather than attach it from the library. The first parameter is the path to the file and the second parameter, set to `false` here, is a flag to indicate whether the file is streaming. There is more on this method later, but in the example shown the file is not streaming. There are a number of event handlers associated with the `Sound` object that you should also be aware of:

- `onLoad` — An event handler that is invoked when the `.mp3` file has loaded or failed to load
- `onSoundComplete` — An event handler that is invoked when an MP3 file finishes playing
- `onID3` — An event handler that is invoked when ID3 information is returned by the file being played

In the code listing, you can see the event handler `onLoadHandler()` is delegated to the `onLoad()` event. If the load of the file is successful, the sound object starts the track at the beginning through the start method. Like the `NetStream` class, the `Sound` class has two methods that can aid in monitoring the data loaded into the application and the file size of the media:

- `getBytesLoaded()`
- `getBytesTotal()`

The `onSoundComplete()` event handler is invoked when the `.mp3` file reaches the end (see Listing 7-2).

www.it-ebooks.info



LISTING 7-2: Using onSoundComplete to handle when a sound file has completed

```
import mx.utils.Delegate;

var sound:Sound = new Sound();
sound.onLoad = Delegate.create(this, onLoadHandler);
sound.onSoundComplete = Delegate.create(this, onSoundCompleteHandler);
sound.loadSound("media/track1.mp3", false);

function onLoadHandler(hasLoaded:Boolean){
      if(hasLoaded){
            trace("Sound file has loaded! ");
            trace("Start it from the beginning! ");
            sound.start(0, 1);
      } else {
            trace("Sound file has not loaded! ");
      }
}

function onSoundCompleteHandler() {
      trace("Sound file has ended!");
}
```

Obtaining ID3 Information File

As soon as an `.mp3` file loads, it will invoke the `onID3` event handler.

The ID3 information can prove to be a very useful asset in a media application. There are two versions of ID3 in use: version 1 and version 2. Files that adhere to the ID3v1.1 standard can contain the following information:

- Song title
- Artist
- Album
- Year
- Comment
- Genre
- Album track (not present in ID3v1.0)

Files that adhere to the ID3v2.3 standard can contain information included in the ID3v1.1 as well as numerous other properties, including the file size, publisher, file length, and even information on how many times a file has been played through a play counter. At the time of writing, ID3v2.3 is the current informal standard of the tagging system. The properties of the ID3 tags are more commonly known as frames of the file. A more comprehensive list of these frames can be found at the ID3 Web site at (ID3, `http://www.id3.org/`).

www.it-ebooks.info

Listing 7-3 illustrates the `onID3` event being handled:



LISTING 7-3: Using onID3 to handle when a ID3 data has been retrieved

```
import mx.utils.Delegate;

var sound:Sound = new Sound();
sound.onLoad = Delegate.create(this, onLoadHandler);
sound.onSoundComplete = Delegate.create(this, onSoundCompleteHandler);
sound.onID3 = Delegate.create(this, onID3Complete);

sound.loadSound("media/track1.mp3", false);

function onLoadHandler(hasLoaded:Boolean){
      if(hasLoaded){
            trace("Sound file has loaded!");
            trace("Start it from the beginning!");
            sound.start(0, 1);
      } else {
            trace("Sound file has not loaded!");
      }
}

function onSoundCompleteHandler() {
      trace("Sound file has ended!");
}

function onID3Complete(hasID3:Boolean) {
      for (var propertyName in sound.id3) {
            trace(propertyName + " = " + sound.id3[propertyName]);
      }
}
```



ID3 is a tagging system widely used in various media software applications. The system was originally pioneered by Erik Kemp as a way to include more information in `.mp3` *files, such as the artist and track name, in addition to the actual* `.mp3` *audio track.*

The `onID3()` event handler essentially returns meta data, which is contained within the file, and as with `.flv` files, MP3s can be encoded with information other than the media track. In this example, when `onID3` event is invoked, it writes the `id3` object to the `Sound` object instance and, like the `onLoad` event handler, indicates whether the invocation is successful or not. AS2 has predefined parameters for the `id3` object. The reference is given in the appendix.

Controlling the Audio for Video

The `Video` object only allows you to interact with the video properties and methods. To interact with sound, remember, you need to use an instance of the `Sound` object. To control the volume of a video, you supply the `Sound` object with a reference to the video object container.

www.it-ebooks.info

The `Sound` class has numerous functions that you can use to manipulate the audio of the sound objects:

- `getPan()` and `setPan()` — Methods that can control the volume of the left and right channels
- `getTransform()` and `setTransform()` — Methods that can manipulate the depth of balance of the left and right channels
- `getVolume()` and `setVolume()` — Methods that are responsible for retrieving and setting the volume level

If you want to control the volume of a video clip being played, you need to use a `NetStream` object, attaching it to a MovieClip sound container, using the `attachAudio()` method. You then instantiate a new `Sound` object, using a reference to the sound container. Then, you can call the `setVolume()` method of the sound object with a button click:




```
import mx.utils.Delegate;

var depth:Number = this.getNextHighestDepth();
var videoContainer:MovieClip = this.attachMovie("videoObj", "videoObj", depth);

var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);

var video:Video = videoContainer. video;
video.attachVideo(ns);

var sDepth:Number = this.getNextHighestDepth();
var soundContainer:MovieClip = this.createEmptyMovieClip("soundObj", sDepth);
soundContainer.attachAudio(ns);
var sound:Sound = new Sound(soundContainer);

ns.play("media/modernWarfare2.flv");

soundOff_btn.onRelease = Delegate.create(this, onMute);

function onMute(){
      sound.setVolume(0);
}

soundOn_btn.onRelease = Delegate.create(this, onUnmute);

function onUnmute(){
      sound.setVolume(100);
}
```

code snippet chapter7_videoAudio.fla

Here, you can see that `soundContainer` is the MovieClip that contains the `NetStream` object, which is then passed as a reference for the `Sound` object instantiation, rather than the linkage ID from the library. Using the `setVolume()` method of the sound object, you can provide a number indicating the level of volume desired. The value range is between 0 and 100.

www.it-ebooks.info

STREAMING AUDIO AND VIDEO

From here on in, you need to use Flash Media Server (FMS). If necessary, you can download a developer copy from the Adobe Web site at `www.adobe.com`.

When you've installed FMS, navigate to the applications directory. There you will see two folders: vod and live. The `<installation>/applications/vod` folder path is the built-in Video on Demand (VOD) service created by Adobe that allows you to stream your media to Flash. It is set up as part of the default installation of FMS. By default, there is a media folder in the vod folder path. This is where the physical files and content for the service should reside. The `/live` application folder is for streaming live.

For streaming media, the `/vod` path can be used to stream both FLV video and MP3 audio to the Flash Lite player. In the `.zip` file accompanying this chapter, transfer the files from the media folder into the `/vod` location. This is where all the files referenced in the Real Time Messaging Protocol (RTMP) URLs will be located.

Unlike version 2.0 of the Flash Lite player, Flash Lite 3.1 supports video streaming from a Flash Media Server over RTMP. The benefits of this will be discussed later. Using Flash Lite 3.*x*, your Flash Videos files (`.flv`, FLV) can be rendered directly by the player. The player uses the On2 and Sorenson codecs, which are optimized for mobile devices streaming video data over an RTMP connection to a Flash Media Server.



Real Time Messaging Protocol Tunnel (RTMPT) and Real Time Messaging Protocol Secure (RTMPS) connections are not supported, nor are multiple connections.

Streaming FLV Video

There are two main differences you need to note between the AS2 code for progressive download playback, already discussed, and the streaming:

1. Instead of making a `null` connection via the `connect()` method of the `NetConnection` object, you need to supply the IP address of the FMS through the Real-Time Messaging Protocol (RTMP). For this chapter you will use the IP address "127.0.0.1:1935".
2. When using the `play()` method of the `NetStatus` object you drop the FLV file extension `.flv`.

Here's a code snippet that shows the connection to the `.flv` file `modernWarfare2.flv`.




```
fscommand2("FullScreen", true);

var depth:Number = this.getNextHighestDepth()
var videoContainer:MovieClip = this.attachMovie("videoObj", "videoObj", depth);

var nc:NetConnection = new NetConnection();
```

www.it-ebooks.info

```
nc.connect("rtmp://127.0.0.1:1935/vod");

var ns:NetStream = new NetStream(nc);

var video:Video = videoContainer.video;
video.attachVideo(ns);

ns.play("modernWarfare2");
```

code snippet chapter7_videoStreaming.fla



You need to specify the IP address; using `rtmp://localhost/vod` *will not work for Flash Lite 3.0.*

Streaming MP3 Audio

Similarly, when streaming `.mp3` files via FMS, you need to modify the `play()` method of the `NetStatus` object to include the name of the MP3 file minus the `.mp3` file extension and pre-append `mp3:` to the file name string (See Listing 7-4):



LISTING 7-4: Streaming an mp3 file from FMS

```
var nc:NetConnection = new NetConnection();
nc.connect("rtmp://127.0.0.1/vod");

var ns:NetStream = new NetStream(nc);
var sDepth:Number = this.getNextHighestDepth();
var soundContainer:MovieClip = this.createEmptyMovieClip("soundObj", sDepth);
soundContainer.attachAudio(ns);

var sound:Sound = new Sound(soundContainer);

ns.play("mp3:track1");
```

As shown in Table 7-1, there are various types of RTMP-based protocols.

TABLE 7-1: Types of RTMP-based protocols

PROTOCOL	RTMP PROTOCOLS
RTMP	Real-Time Messaging Protocol
RTMPT	Real-Time Messaging Protocol Tunneled over HTTP
RTMPE	128-bit encrypted Real-Time Messaging Protocol
RTMPTE	128-bit encrypted Real-Time Messaging Protocol Tunneled over HTTP
RTMPS	Real-Time Messaging Protocol over SSL

www.it-ebooks.info

System.capabilities Revisited

In Chapter 3 you saw how you could use the `System` class to detect the capabilities of a device via `System.capabilities`. The following list details a list of values that can be retrieved to detect particular capabilities relating to media; each value returns a Boolean value, either true or false:

- `System.VideoMIMETypes` — Returns an array of Video MIME types supported on the device
- `System.capabilities.hasEmbeddedVideo` — Is the device capable of embedding video?
- `System.capabilities.hasStreamingAudio` — Is the device capable of streaming audio?
- `System.capabilities.hasStreamingVideo` — Is the device capable of streaming video?
- `System.capabilities.hasVideoEncoder` — Is the device capable of encoding video?

Having covered numerous aspects of video and sound, you'll now take a look at Media Console created for this chapter.

BUILDING THE APPLICATION

In this section you'll explore the code behind the Media Console application, building on the knowledge you've learned this far.

You'll look at the following areas:

- The `.fla` File
- Defining ApplicationFacade
- Creating a Model for the Media Console
- Creating the Media Console's Controller
- Exploring the View's Mediators

First, you should assume that the devices the application runs on will be able to play back audio or video. Hence for the `System.capabilities` object, the `hasEmbeddedVideo`, `hasAudio`, `hasMP3`, `hasVideoEncoder`, and `hasAudioEncoder` properties should be designated as `true`. If any of these retrieved values comes back as `false`, then an appropriate message should be displayed to the user.

The .fla File

Open the `chapter7_MediaConsole.fla` file in Flash CS4. In the Library panel for the application, you'll see that there are several assets, all of which are used in the application, including `SoftKeys`, `StatusBar`, `Title`, and `ListTwoRowWithIcon` components covered in Chapter 4. Then there's a `Video` component, and two containers for video and sound called `videoContainer` and `soundContainer`, respectively.

On the first frame of the timeline, you'll see the script in Listing 7-5, which starts the application.

www.it-ebooks.info



LISTING 7-5: Starting the application

```
import com.wrox.chapter7.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
app.startup(this);
```

Defining ApplicationFacade

Now you'll define `ApplicationFacade`.

Declaring the Notification Names

For the media console, the `ApplicationFacade` needs three methods, defining: `getInstance()`, `initializeController()`, and `startup()`. The class also requires several notification names to be defined. In the following list, all of the notification names used for the application are defined, with their corresponding commands mentioned where applicable:

- `STARTUP` — Mapped to the `StartupCommand` and is responsible for initializing the model and view
- `LOAD_PLAYLIST` — Mapped to `LoadPlaylistCommand` and is responsible for initializing the loading of `audioPlaylist.xml`
- `INIT_SOUND` — Mapped to `InitSoundCommand` and is responsible for initializing the sound
- `SELECT_PLAYLIST_ITEM` — Mapped to `SelectPlaylistItemCommand` and is responsible for selection of playlist items
- `TOGGLE_SOUND` — Mapped to `ToggleSoundCommand` and handles the call to mute and unmute sound
- `PLAY` — Mapped to `PlayCommand` and is responsible for handling the selection of playlist items
- `PAUSE` — Mapped to `PauseCommand` and is responsible for selection of playlist items
- `RESTART` — Mapped to the `RestartCommand`
- `PLAY_NEXT` — Mapped to `PlayNextCommand` and is responsible for playing the next item in the playlist
- `PLAY_PREVIOUS` — Mapped to `PlayPreviousCommand` and is responsible for playing the previous item in the playlist

Next, you can follow the steps to create the `ApplicationFacade`.

1. Begin by creating the basic shell of the `ApplicationFacade` class (see Listing 7-6).



LISTING 7-6: Creating the PlaylistProxy for the Media Console

```
import com.wrox.chapter7.model.*;
import com.wrox.chapter7.view.*;
import com.wrox.chapter7.controller.*;
```

continues

www.it-ebooks.info

LISTING 7-6 *(continued)*

```
import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.facade.*;

class com.wrox.chapter7.ApplicationFacade extends Facade implements IFacade {

}
```

2. Create the public `getInstance()` method. This should return an `instance` of `ApplicationFacade` (see Listing 7-7).



LISTING 7-7: Creating the getInstance method for ApplicationFacade

```
import com.wrox.chapter7.model.*;
import com.wrox.chapter7.view.*;
import com.wrox.chapter7.controller.*;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.facade.*;

class com.wrox.chapter7.ApplicationFacade extends Facade implements IFacade {

      public static function getInstance():ApplicationFacade {

            if (instance == null) {
                  instance = new ApplicationFacade();
            }

            return ApplicationFacade(instance);
      }

}
```

3. Next, define the notification names (see Listing 7-8).



LISTING 7-8: Defining the notification names in ApplicationFacade

```
import com.wrox.chapter7.model.*;
import com.wrox.chapter7.view.*;
import com.wrox.chapter7.controller.*;

import org.puremvc.as2.interfaces.*;
import org.puremvc.as2.patterns.facade.*;

class com.wrox.chapter7.ApplicationFacade extends Facade implements IFacade {

      public static var STARTUP:String = "startUp";
      public static var QUIT:String = "quit";
      public static var LOAD_PLAYLIST:String = "loadPlaylist";
      public static var SHOW_PLAYLIST:String = "showPlaylist";
```

www.it-ebooks.info

```
        public static var NET_CONNECTION_SUCCESS:String = "ncSuccess";
        public static var AUDIO_STREAM_CREATED:String = "audioStreamCreated";
        public static var VIDEO_STREAM_CREATED:String = "videoStreamCreated";
        public static var NET_CONNECTION_CLOSED:String = "ncClosed";
        public static var INIT_SOUND:String = "initSound";
        public static var ATTACH_SOUND:String = "attachSound";
        public static var TOGGLE_SOUND:String = "toggleSound";
        public static var SHOW_MEDIA:String = "showMedia";
        public static var HIDE_MEDIA:String = "hideMedia";
        public static var SELECT_PLAYLIST_ITEM:String = "selectPlaylistItem";
        public static var PLAY:String = "play";
        public static var PAUSE:String = "pause";
        public static var RESTART:String = "restart";
        public static var PLAY_PREVIOUS:String = "playPrevious";
        public static var PLAY_NEXT:String = "playNext";
        public static var PLAYLIST_ITEM_CHANGED:String = "playlistItemChanged";
        public static var MEDIA_TIME_UPDATED:String = "mediaTimeUpdated";
        public static var UPDATE_SOFT_KEYS:String = "updateSoftKeys";

        public static function getInstance():ApplicationFacade {

              if (instance == null) {
                    instance = new ApplicationFacade();
              }

              return ApplicationFacade(instance);
        }

}
```

4. Under the `getInstance()` method, define the `startup()` method. This needs to receive the reference to the application `app` in the parameters. Ensure that the `app` reference is sent in the body of `START_UP` notification (see Listing 7-9).



LISTING 7-9: The startup method for ApplicationFacade

```
public static function getInstance():ApplicationFacade {
      if (instance == null) {
            instance = new ApplicationFacade();
      }
      return ApplicationFacade(instance);
}

public function startup(app:MovieClip):Void {
     sendNotification(STARTUP, app);
}
```

5. Next, create the `initializeController()` method. Here, you need to register each of the commands mentioned earlier (see Listing 7-10).

www.it-ebooks.info



LISTING 7-10: Defining the initializeController method for ApplicationFacade

```
public function startup(app:MovieClip):Void {
     sendNotification(STARTUP, app);
}

private function initializeController():Void {
      super.initializeController();
      registerCommand(STARTUP, StartupCommand);
      registerCommand(LOAD_PLAYLIST, LoadPlaylistCommand);
      registerCommand(SHOW_PLAYLIST, ShowPlaylistCommand);
      registerCommand(TOGGLE_SOUND, ToggleSoundCommand);
      registerCommand(SHOW_MEDIA, ShowMediaCommand);
      registerCommand(HIDE_MEDIA, HideMediaCommand);
      registerCommand(PLAY, PlayCommand);
      registerCommand(PAUSE, PauseCommand);
      registerCommand(RESTART, RestartCommand);
      registerCommand(PLAY_NEXT, PlayNextCommand);
      registerCommand(PLAY_PREVIOUS, PlayPreviousCommand);
}
```

You've now gone through each aspect of the `ApplicationFacade`. Be sure to take note of each of the commands and notification names defined here, as these will be referenced throughout the remainder of building the Media Console. Next, take a look at the model for the Media Console.

Creating a Model for the Media Console

In this section you'll examine the model defined for the application (see Figure 7-1).

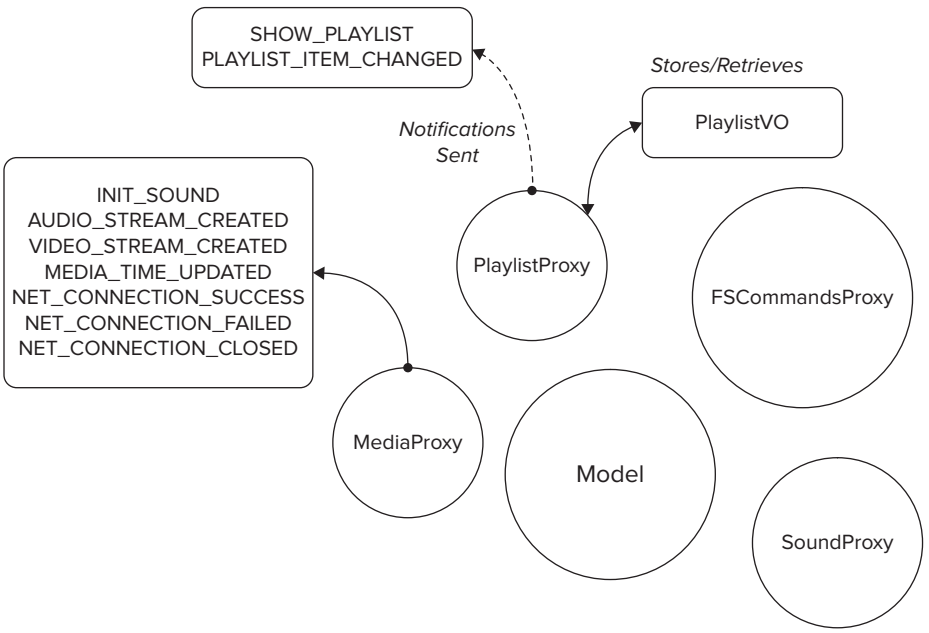


FIGURE 7-1

www.it-ebooks.info

In Figure 7-1, you can see three new proxy classes that are used for the Media console application:

- `PlaylistProxy` — To manage the playlist data used
- `MediaProxy` — To manage the streaming of media
- `SoundProxy` — To handle sound

The Project panel shown in Figure 7-2 illustrates the file system layout for the Media Console, highlighting each of the files in the model directory.

Before you move on to the core of the application, have a look at some of the external resources used in the Media Console.



FIGURE 7-2

The Audio Playlist

Central to the application is the audio data you'll be using to reference and access media. Continuing with the usage of XML, the Media Console uses the `audioPlaylist.xml` file as external data to the MP3s on the server. This file contains a reference to the "artist," the track "title," "url" path, and "type" of media, as shown in the following code snippet:

Available for download on Wrox.com

```
<playlist>
<item>
      <title>New Friends</title>
      <artist>Loudog</artist>
      <duration>05:27</duration>
      <url>rtmp://127.0.0.1/vod/track1</url>
      <type>mp3</type>
</item>
  <item>
      <title>Wadidyusay?</title>
      <artist>Zap Mama</artist>
      <url>rtmp://127.0.0.1/vod/track2</url>
      <duration>03:16</duration>
      <type>mp3</type>
</item>
<item>
      <title>Do It Over</title>
      <artist>Amelie</artist>
      <url>rtmp://127.0.0.1/vod/track3</url>
      <duration>03:05</duration>
      <type>mp3</type>
</item>
<item>
      <title>Climate Change</title>
      <artist>Floppy Dee</artist>
      <url>rtmp://127.0.0.1/vod/track4</url>
      <duration>01:56</duration>
      <type>mp3</type>
</item>
```

www.it-ebooks.info

```
<item>
      <title>Revolution Now</title>
      <url>rtmp://127.0.0.1/vod/track5</url>
      <type>mp3</type>
      <artist>JuaKali
      <duration>02:50</duration>
</item>
<item>
      <title>Epic</title>
      <url>rtmp://127.0.0.1/vod/track6</url>
      <type>mp3</type>
      <artist>djbouly</artist>
      <duration>06:19</duration>
</item>
<item>
      <title>Satellite</title>
      <url>rtmp://127.0.0.1/vod/track7</url>
      <type>mp3</type>
      <artist>major Major</artist>
      <duration>03:09</duration>
</item>
<item>
      <title>Effortless</title>
      <url>rtmp://127.0.0.1/vod/track8</url>
      <type>mp3</type>
      <artist>Josh Woodward</artist>
      <duration>03:50</duration>
</item>
<item>
      <title>WATER (Santoor by Manish Vyas)</title>
      <url>rtmp://127.0.0.1/vod/track9</url>
      <type>mp3</type>
      <artist>SaReGaMa</artist>
      <duration>07:31</duration>
</item>
</playlist>
```

code snippet audioPlaylist.xml

PlaylistItemVO

The data model for the Media Console will use a `PlaylistItemVO` class, which contains a reference to each of the nodes and properties we want to use in the `XML`.

The full `PlaylistItemVO` is shown in Listing 7-11.



LISTING 7-11: The PlaylistItemVO class

```
class com.wrox.chapter7.model.vo.PlaylistItemVO {

      public var artist:String;
      public var duration:String;
      public var title:String;
```

www.it-ebooks.info

```
        public var type:String;
        public var url:String;

        public function PlaylistItemVO() {

        }
    }
```

Shortly, you'll take a look at the `PlaylistProxy`, which is responsible for parsing the `audioPlaylist.xml` file and distributing `PlaylistItemVO` instances.

Retrieving the Playlist

Having looked at the `audioPlaylist.xml` file and the `PlaylistItemVO` class, in this section you'll cover the `PlaylistProxy`, the class tasked with loading the `audioPlaylist.xml` file, and storing the data for retrieval, which is an array of `PlaylistItemVO` instances.

The `PlaylistProxy` will utilize a loader class `PlaylistLoader` and parser class `PlaylistParser`; you'll go through these shortly. The `PlaylistProxy` will have seven public methods defined. These will be accessed via several commands in the application's framework, which you'll take a look at later.

- `loadPlaylist()` — To load the playlist
- `setSelectedIndex()` — Sets the index position of the chosen playlist item
- `getServer()` — Gets the server path of the current playlist item
- `getFile()` — Gets the file path of the current playlist item
- `getTitle()` — Gets the title of the selected playlist item
- `getArtist()` — Gets the artist of the selected playlist item
- `getDuration()` — Get the duration of the selected playlist item
- `setNextItem()` — Sets the next item in the playlist
- `setPreviousItem()` — Sets the previous item in the playlist

In the `loadPlaylist()` method, an instance of `PlaylistLoader` is used to retrieve `audioPlaylist.xml` and then the instance of `PlaylistParser` is used to parse and return each of the items in the list.

PlaylistLoader.as

The `loadPlaylist()` method for the `PlaylistProxy` invokes the `init()` method of `PlaylistLoader`, which is where a familiar call to load XML is made (see Listing 7-12).



LISTING 7-12: The PlaylistLoader class for the Media Console

```
import mx.events.EventDispatcher;
import mx.utils.Delegate;

class com.wrox.chapter7.model.loaders.PlaylistLoader {
```

continues

www.it-ebooks.info

LISTING 7-12 *(continued)*

```
        public var dispatchEvent:Function;
        public var addEventListener:Function;
        public var removeEventListener:Function;

        private var xml:XML;

        public function PlaylistLoader() {
              EventDispatcher.initialize(this);
        }

        public function init(url:String) {
              xml = new XML();
              xml.ignoreWhite = true;
              xml.onLoad = Delegate.create(this, onRequestComplete);
              xml.load(url);
        }

        private function onRequestComplete(success:Boolean):Void {
              if (success) {
                    dispatchEvent({target:this, type:"onComplete"});
              } else {
                    dispatchEvent({target:this, type:"onFailed"});
              }
        }

        public function getData():XML {
              return xml;
        }
}
```

PlaylistParser.as

The `playlist` parser returns an array of playlist items via the public `getPlaylist()` method. Each node of `audioPlaylist.xml` is parsed into the `PlaylistItemVO`. The `PlaylistItemVO` has five values: `artist`, which is the artist for the track; `title`, which is a reference to the track name; `url`, the path to the file location; `type`, which defines whether or not the track is an MP3, and `duration`, which gives the length of the track. The type value provides some flexibility for handling other types of media (See Listing 7-13).



LISTING 7-13: The parseXML, parseItemXMLNode and getPlaylist methods for PlaylistParser

```
public function parseXML(xml:XML):Void {

        var xmlNode:XMLNode = xml.firstChild;

        playlist = new Array();

        var xmlNodeList:Array = XPathAPI.selectNodeList(xmlNode,  "playlist/item");
```

www.it-ebooks.info

```
            for( var i:Number=0; i < xmlNodeList.length; i++ ) {
                  playlist.push( parseItemXMLNode(xmlNodeList[i]) );
            }

            dispatchEvent( { target:this, type:"onParseComplete" } );
      }

      private function parseItemXMLNode(node:XMLNode):Object {

            var vo:PlaylistItemVO = new PlaylistItemVO();
            var childNode:XMLNode;

            childNode = XPathAPI.selectSingleNode(node, "item/artist");
            vo.artist = childNode.firstChild.nodeValue;

            childNode = XPathAPI.selectSingleNode(node, "item/duration");
            vo.duration = childNode.firstChild.nodeValue;

            childNode = XPathAPI.selectSingleNode(node, "item/title");
            vo.title = childNode.firstChild.nodeValue;

            childNode = XPathAPI.selectSingleNode(node, "item/url");
            vo.url = childNode.firstChild.nodeValue;

            childNode = XPathAPI.selectSingleNode(node, "item/type");
            vo.type = childNode.firstChild.nodeValue;

            return vo;
      }

      public function getPlaylist():Array {
            return playlist;
      }
```

PlaylistProxy.as

Next, we'll take a look at creating the `PlaylistProxy`.

1. Begin by creating the skeleton for `PlaylistProxy`. Ensure that you include the static `NAME` variable for the Proxy and the `url` variable as a reference to the `audioPlaylist.xml` file (see Listing 7-14).



LISTING 7-14: Creating the PlaylistProxy for Media Console

```
import com.wrox.chapter7.ApplicationFacade;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;
import mx.utils.Delegate;

class com.wrox.chapter7.model.PlaylistProxy extends Proxy implements IProxy {

      public static var NAME:String = "PlaylistProxy";
```

continues

www.it-ebooks.info

LISTING 7-14 *(continued)*

```
        private var url:String = "audioPlaylist.xml";

        public function PlaylistProxy(data:Object){
              super(NAME, data);
        }

        public function loadPlaylist():Void { }

        private function onLoadComplete(success:Boolean):Void { }

        private function onParseComplete():Void { }

        public function setSelectedIndex(index:Number):Void { }

        public function setNextItem():Void { }

        public function setPreviousItem():Void { }

        public function getArtist():String { }

        public function getTitle():String { }

        public function getUrl():String { }

        public function getType():String { }

        public function getDuration():String { }

        public function getServer():String { }

        public function getFile():String { }

        private function isRTMP(url:String):Boolean { }

        private function isMP3(str:String):Boolean { }
}
```

2. Next, import `PlaylistLoader`, `PlaylistParser`, and `PlaylistItemVO`, then create three class-level variable instances for each of them. Then, create stubs for `loadPlaylist()`, `onLoadComplete()`, and `onParseComplete()` methods (see Listing 7-15).



LISTING 7-15: Creating the PlaylistProxy for Media Console

```
import com.wrox.chapter7.ApplicationFacade;
import com.wrox.chapter7.model.loaders.PlaylistLoader;
import com.wrox.chapter7.model.parsers.PlaylistParser;
import com.wrox.chapter7.model.vo.PlaylistItemVO;
import org.puremvc.as2.interfaces.IProxy;
```

www.it-ebooks.info

```
import org.puremvc.as2.patterns.proxy.Proxy;
import mx.utils.Delegate;

class com.wrox.chapter7.model.PlaylistProxy extends Proxy implements IProxy {

      public static var NAME:String = "PlaylistProxy";

      private var playlistLoader:PlaylistLoader;
      private var playlistParser:PlaylistParser;
      private var selectedItem:PlaylistItemVO;
      private var url:String = "audioPlaylist.xml";

      public function PlaylistProxy(data:Object) {
            super(NAME, data);
      }

      public function loadPlaylist():Void {}

      private function onLoadComplete(success:Boolean):Void {}

      private function onParseComplete():Void {}

      public function setSelectedIndex(index:Number):Void { }

      public function setNextItem():Void { }

      public function setPreviousItem():Void { }

      public function getArtist():String { }

      public function getTitle():String { }

      public function getUrl():String { }

      public function getType():String { }

      public function getDuration():String { }

      public function getServer():String { }

      public function getFile():String { }

      private function isRTMP(url:String):Boolean { }

      private function isMP3(str:String):Boolean { }

}
```

3. Complete the `loadPlaylist()` method. Delegate the `onComplete` event to the `onLoadComplete()` method and make a call to load the `playlistUrl` through the `playlistLoader.init()` method (see Listing 7-16).

www.it-ebooks.info



LISTING 7-16: The loadPlaylist method for PlaylistProxy

```
public function loadPlaylist():Void {
      playlistLoader = new PlaylistLoader();
      playlistLoader.addEventListener("onComplete", Delegate.create(this,
            onLoadComplete));
      playlistLoader.init(playlistUrl);
}
```

4. Complete the `onLoadComplete()` method. Once the XML is loaded by `playlistLoader`, the `onLoadComplete()` method is called and the `data` object in `playlistLoader` is passed as parameter to the `playlistParser.parseXML()` method (see Listing 7-17).



LISTING 7-17: The onLoadComplete method for PlaylistProxy

```
private function onLoadComplete(success:Boolean):Void {
      playlistParser = new PlaylistParser();
      playlistParser.addEventListener("onParseComplete", Delegate.create(this,
            onParseComplete ));
      playlistParser.parseXML( playlistLoader.getData() );
}
```

5. Next, complete the `onParseComplete()` method. In `onParseComplete()`, you send the `SHOW_PLAYLIST` notification. Here, create two arrays, `titles` and `artists`, to hold the data retrieved from the playlist. These are then assigned to a playlist object's properties, `titleArr` and `artistArr`. The playlist object, `playlistObj`, is then sent in the body of the notification (see Listing 7-18).



LISTING 7-18: The onParseComplete method for PlaylistProxy

```
private function onParseComplete():Void {
      setData( playlistParser.getPlaylist() );

      var titles:Array = [];
      var artists:Array = [];

      for (var i:Number = 0; i < data.length; i++ ) {
            titles.push(PlaylistItemVO(data[i]).title);
            artists.push(PlaylistItemVO(data[i]).title);
      }

      var playlistObj:Object = { titleArr:titles, artistArr:artists };

      sendNotification(ApplicationFacade.SHOW_PLAYLIST, playlistObj);
}
```

6. Next, add the `selectedIndex` variable (see Listing 7-19).

www.it-ebooks.info



LISTING 7-19: Declaring the selectedIndex variable in PlaylistProxy

```
private var playlistLoader:PlaylistLoader;
private var playlistParser:PlaylistParser;
private var selectedItem:PlaylistItemVO;
private var selectedIndex:Number;
private var playlistUrl:String = "audioPlaylist.xml";
```

7. Next, create the `setSelectedIndex()`, `setNextItem()`, and `setPreviousItem()` methods. For `setSelectedIndex()`, you need to set the `selectedIndex` variable to the item selected by the user, whereas `setNextItem()` and `setPreviousItem()` need to send the `PLAYLIST_ITEM_CHANGED` notification, passing a reference to the title and artist parameters in the notification body (see Listing 7-20).



LISTING 7-20: The setSelectedIndex, method for PlaylistProxy

```
public function setSelectedIndex(index:Number):Void {
      selectedIndex = index;
      selectedItem = new PlaylistItemVO();
      selectedItem = PlaylistItemVO(data[index]);
}

public function setNextItem():Void {
      selectedIndex++;

      if (selectedIndex > data.length) {
            selectedIndex = 0;
      }

      selectedItem = new PlaylistItemVO();
      selectedItem = PlaylistItemVO(data[selectedIndex]);

}

public function setPreviousItem():Void {
      selectedIndex--;

      if (selectedIndex <= -1) {
            selectedIndex = Array(data).length-1;
      }

      selectedItem = new PlaylistItemVO();
      selectedItem = PlaylistItemVO(data[selectedIndex]);

      sendNotification(ApplicationFacade.PLAYLIST_ITEM_CHANGED);
}
```

8. Complete the methods to return the `artist`, `title`, `url`, `type`, and `duration` properties. These methods should return the values from `selectedItem`, which has the data type of `PlaylistItemVO` (see Listing 7-21).

www.it-ebooks.info



LISTING 7-21: The getArtist, getTitle, getUrl, getType, and getDuration methods for PlaylistProxy

```
public function getArtist():String {
      return selectedItem.artist;
}

public function getTitle():String {
      return selectedItem.title;
}

public function getUrl():String {
      return selectedItem.url;
}

public function getType():String {
      return selectedItem.type;
}

public function getDuration():String {
      return selectedItem.duration;
}
```

9. Next, create the `isRTMP()` and `isMP3()` methods. These are private methods which are used internally by the class:

   - `isRTMP()` — Used to determine whether or not a playlist item should be played via RTMP and returns `true` or `false`
   - `isMP3()` — Used to determine whether or not a playlist item is an `.mp3` file and returns `true` or `false` (see Listing 7-22)



LISTING 7-22: The isRTMP and isMP3 method for PlaylistProxy

```
private function isRTMP(url:String):Boolean {
      var rtmp:Boolean = false;
      if (url.indexOf("rtmp") != -1) rtmp = true;

      return rtmp;
}

private function isMP3(str:String):Boolean {
      var mp3:Boolean = false;
      if (str.indexOf("mp3") != -1) mp3 = true;

      return mp3;
}
```

10. Next create the `getServer()` and `getFile()` methods. The `getServer()` method needs to return the server's address. This is required for the `NetConnection` class, which will be covered in `MediaProxy`. Determining whether the playlist item requires RTMP is done

www.it-ebooks.info

here also. Similarly, with `getFile()` you retrieve the file path used for the `NetStream` (see Listing 7-23). Here, you utilize the `isRTMP()` and `isMP3()` methods.



LISTING 7-23: The getServer and getFile methods for PlaylistProxy

```
public function getServer():String {
      var server:String = "null";
      var itemUrl = selectedItem.url;

      if (isRTMP(itemUrl)) {
            server = itemUrl.substring(0, itemUrl.lastIndexOf("/"));
      }

      return server;
}

public function getFile():String {
      var file:String = selectedItem.url;

      if (isRTMP(selectedItem.url)) {
            file = file.substring( Number(file.lastIndexOf("/") + 1), file.length);

            if ( isMP3(selectedItem.url) || isMP3(selectedItem.type) ) {
                  file = "mp3:" + file;
            }
      }

      return file;
}
```

This completes the `PlaylistProxy`. Next, you'll take a look at `MediaProxy`, which is at the core of the media console.

Handling Media Playback Requests

Here you'll learn how to handle media playback requests.

MediaProxy.as

The `MediaProxy` class will initialize the calls to play media through using the `NetConnection` and `NetStream` classes and will need to have the following methods defined:

- `makeConnection()` — Used to make a connection via the `NetConnection` class
- `makeStream()` — Used to create the stream
- `onConnectionStatus()` — To handle the net connection status events
- `onStreamStatus()` — Used to handle the net stream status events
- `onMetaData()` — Used to handle meta data retrieval
- `closeConnection()` — Used to close a net connection

www.it-ebooks.info

- `checkTime()` — Used to check the time of the net stream while the media is playing
- `getNetConnection()` — Used to retrieve a net connection
- `getNetStream()` — Used to retrieve the current net stream
- `init()` — Used to initialize a call to the `MediaProxy`
- `initInterval()` — Used to initialize an update to the net stream application on the media time
- `playStream()` — Used to start media playback
- `playFrom()` — Used to start media from a specific position
- `pause()` — Used to pause media
- `resume()` — Used to resume media playback

Follow the next steps to create the `MediaProxy` class.

1. Begin by creating the skeleton for the proxy; include all the stubs for the methods, and add two private variables for the `NetConnection` and `NetStream` (see Listing 7-24).



LISTING 7-24: Creating the MediaProxy for the Media Console

```
import com.wrox.chapter7.ApplicationFacade;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter7.model.MediaProxy extends Proxy implements IProxy {

      public static var NAME:String = "MediaProxy";

      private var nc:NetConnection;
      private var ns:NetStream;

      public function MediaProxy(data:Object) {
            super(NAME, data);
      }

      public function onRegister():Void { }

      public function init():Void { }

      private function makeConnection():Void {  }

      private function makeStream():Void { }

      private function onConnectionStatus():Void { }

      private function onStreamStatus():Void { }
```

www.it-ebooks.info

```
        private function checkTime():Void { }

        private function initInterval():Void { }

        public function onMetaData():Void { }

        public function closeConnection():Void { }

        public function playFrom():Void { }

        public function playStream():Void { }

        public function pause():Void { }

        public function resume():Void { }

        public function close():Void { }

        public function getNetConnection():NetConnection { }

        public function getNetStream():NetStream { }

}
```

2. Next, add five private `String` variables to the proxy, called `file`, `server`, `media`, `duration`, and `timeInterval`. Then, add two public variables called `isPlaying` and `isPaused` with Boolean data types. The `MediaProxy` will keep a reference to the state of play, through `Boolean` variables `isPlaying` and `isPaused`, to help toggle between calling the `play()` method and calling the `resume()` method. To call the `play()` method, run media from the start, should it be paused and not playing, and resume. If you remember, these values can be retrieved from the `PlaylistProxy` (see Listing 7-25).



LISTING 7-25: Private variables for the MediaProxy

```
private var nc:NetConnection;
private var ns:NetStream;

private var server:String;
private var file:String;
private var media:String;
private var duration:String;
private var timeInterval:String;

public var isPlaying:Boolean;
public var isPaused:Boolean;
```

3. Next, complete the public `init()` method. This should set the values for `server`, `file`, `media`, and `duration` variables. The method should then call the `makeConnection()` method (see Listing 7-26).

www.it-ebooks.info



LISTING 7-26: The init method for MediaProxy

```
public function init(serverPath:String, fileName:String, mediaType:String,
mediaDuration:String):Void {
      server = serverPath;
      file = fileName;
      media = mediaType;
      duration = mediDuration;

      makeConnection();
}
```

4. Update the `closeConnection()` method. This will set the *play state* flags to `false` and close the net stream. It is considered good practice to close a network connection before another is made; the `closeConnection()` method serves this purpose (see Listing 7-27).



LISTING 7-27: The closeConnection method for MediaProxy

```
public function closeConnection():Void {
      if (nc.isConnected && nc != null) {
            isPlaying = false;
            isPaused = false;
            ns.close();
      }
}
```

5. Next complete the `makeConnection()` method. Here, you need to ensure that the current connection is closed before attempting to make a connection. You should note that the method serves two purposes. The first is to determine whether or not the net connection requires a server connection, and the second is to determine if the media type `media` is an MP3. If the media type is an MP3 and there is no server to connect to, you can assume that the `.mp3` file is local. So, here, we send the `INIT_SOUND` notification along with the file path to the `SoundProxy`, which we'll cover shortly (see Listing 7-28).



LISTING 7-28: The makeConnection method for MediaProxy

```
private function makeConnection():Void {
      closeConnection();

      if (server == "null" ) {
            if (media == "mp3") {
                  sendNotification(ApplicationFacade.INIT_SOUND, file);
            } else {
                  nc.connect(null);
            }
      } else {
            nc.connect(server);
      }
}
```

www.it-ebooks.info

6. Next, create the `makeStream()` method. Instantiate a new `NetStream` object, using the net connection defined in `makeConnection()`, and define event handlers for the `onStatus` and `onMetaData` events. Set the `bufferTime` of the net stream to 2 seconds. You then need to play the stream through the `playStream()` method and send the `AUDIO_STREAM_CREATED` notification (see Listing 7-29).



LISTING 7-29: The makeStream method for MediaProxy

```
private function makeStream():Void {
      ns = new NetStream(nc);
      ns.onStatus = Delegate.create(this, onStreamStatus);
      ns.onMetaData = Delegate.create(this, onMetaData);
      ns.setBufferTime(2);

      playStream(file);

      if(media == "mp3"){
            sendNotification(ApplicationFacade.AUDIO_STREAM_CREATED, ns);
      } else {
            sendNotification(ApplicationFacade.VIDEO_STREAM_CREATED, ns);
      }
}
```

7. Next, complete the `onStreamStatus()`, `onMetaData()`, and `onConnectionStatus()` (see Listing 7-30).



LISTING 7-30: The onConnectionStatus, onStreamStatus and onMetaData methods for MediaProxy

```
private function onConnectionStatus(status:Object):Void {
      switch(status.code) {
            case "NetConnection.Connect.Success":
                  makeStream();
                  sendNotification(ApplicationFacade.NET_CONNECTION_SUCCESS);
            break;
            case "NetConnection.Connect.Failed":
            break;
            case "NetConnection.Connect.Rejected":
            break;
            case "NetConnection.Connect.Closed":
            break;
            case "NetConnection.Connect.AppShutdown":
            break;
      }
}

private function onStreamStatus(status:Object):Void {
      switch(status){
            case "NetStream.Play.StreamNotFound":
            break;
```

continues

www.it-ebooks.info

LISTING 7-30 *(continued)*

```
                case "NetStream.Play.Stop":
                break;
                case "NetStream.Play.Reset":
                break;
                case "NetStream.Play.Start":
                break;
                case "NetStream.Unpause.Notify":
                break;
        }
}

private function onMetaData(metaData:Object):Void {
        for (var propertyName:String in metaData) {
                trace(propertyName + " = " + metaData[propertyName]);
        }
}
```

In the `onConnectionStatus()` method, notice that the `NetConnection.Connect.Success` string is captured from the connection's `status.code` property. Once this is retrieved, the `makeStream()` method is called and the `NET_CONNECTION_SUCCESS` notification is sent. There are other status codes that have been defined in the method that haven't been implemented but are listed in the `switch()` statement, including:

- `NetConnection.Connect.Failed` — Used when the net connection fails
- `NetConnection.Connect.Rejected` — Used when the net connection is rejected
- `NetConnection.Connect.Closed` — Used when the net connection is closed
- `NetConnection.Connect.AppShutdown` — Used when the application shuts down

For both the `onStreamStatus()` and `onMetaData()` methods, notice that there is no real implementation being made. The application could be expanded by providing net status information and utilizing the meta data information, and these methods are where the data would be captured.

The stream's status codes are also defined in the `switch()` statement for the `onStreamStatus()` method.

- `NetStream.Play.StreamNotFound` — Used when the stream isn't found
- `NetStream.Play.Stop` — Used when the stream stops playing
- `NetStream.Play.Reset` — Used when the stream is reset
- `NetStream.Play.Start` — Used when the stream is started
- `NetStream.Unpause.Notify` — Used when the stream is "unpaused"

8. Next, create the `playStream()` and `playFrom()` methods. What you are trying to achieve here are two methods that handle the playback of media differently. The `playFrom()`

www.it-ebooks.info

method requires a parameter, which should be the number of seconds into the track that needs to be played, whereas the `playStream()` method uses the `isPaused` flag to determine whether to resume play of a file that has been paused, or play a file from the beginning. The `initInterval()` method should also be called here (see Listing 7-31).



LISTING 7-31: The playFrom and playStream methods for MediaProxy

```
public function playFrom(seconds:Number):Void {
      ns.seek(seconds);
}

public function playStream():Void {
      if (isPaused) {
            resume();
      } else {
            ns.play(file);
            initInterval();
      }
}
```

9. Create the `pause()`, `resume()`, and `close()` methods for the `MediaProxy`, setting the `isPlaying` and `isPaused` to indicate the media play state. In `pause()`, the net stream's `pause()` method should be invoked, along with the `clearInterval()` method so that the `timeInterval` variable is reset. For `resume()`, the `initInterval()` should be called in addition to the net stream's `pause()` method. Then, for `close()`, the net stream's `close()` method should be invoked along with `clearInterval()` (see Listing 7-32).



LISTING 7-32: The pause, resume, and close methods for MediaProxy

```
public function pause():Void {
      isPlaying = false;
      isPaused = true;
      ns.pause(true);
      clearInterval(timeInterval);
}

public function resume():Void {
      isPlaying = true;
      isPaused = false;
      ns.pause(false);
      initInterval();
}

public function close():Void {
      isPlaying = false;
      isPaused = false;
```

continues

www.it-ebooks.info

LISTING 7-32 *(continued)*

```
            ns.close();
            clearInterval(timeInterval);
}
```

10. Next, create the `getNetConnection()` and `getNetStream()` methods to return instances of `nc` and `ns` respectively (see Listing 7-33).



LISTING 7-33: The getNetConnection and getNetStream methods for MediaProxy

```
public function getNetConnection():NetConnection {
            return nc;
}

public function getNetStream():NetStream {
            return ns;
}
```

11. Next, complete the `checkTime()` and `initInterval()` methods. The `checkTime()` method needs to send the `MEDIA_TIME_UPDATED` notification with an array in the body of the notification containing the current time of the `NetStream` object `ns`, and the `duration` of the media, which is represented by the total number of seconds. The `initInterval()` method should invoke `setInterval()`, which should be set to call the `checkTime()` function when called (see Listing 7-34).



LISTING 7-34: The checkTime and initInterval methods for MediaProxy

```
private function checkTime():Void {
            var miniutesToSeconds:Number = Number(duration.split(":")[0] * 60);
            var totalSeconds:Number = miniutesToSeconds + Number(duration.split(":")[1]);
            var times:Array = [ns.time, totalSeconds];

            if (ns.time <= totalSeconds) {
                  sendNotification(ApplicationFacade.MEDIA_TIME_UPDATED, times)
            }
}

private function initInterval():Void {
            timeInterval = setInterval(Delegate.create(this, checkTime), 900);
}
```

12. Finally, complete the `onRegister()` method. Here's where the `NetConnection` object, `nc`, should be instantiated and where the `onStatus` event handler assigned (see Listing 7-35).

www.it-ebooks.info



LISTING 7-35: The onRegister method for MediaProxy

```
public function onRegister():Void {
      nc = new NetConnection();
      nc.onStatus = Delegate.create(this, onConnectionStatus);
}
```

You've now covered each aspect of `MediaProxy`. Next, you'll take a look at creating `SoundProxy`.

Handling Audio

Here you'll learn how to handle audio.

SoundProxy.as

In this section you'll take a look at `SoundProxy`, which has the following methods defined:

- `loadSound()` — Load a sound file
- `attachSound()` — Attach a sound to a container
- `mute()` — Set the sound volume to zero
- `unmute()` — Set the sound back to the previous volume
- `start()` — Start the playback of the sound
- `onRegister()` — Used to instantiate a `Sound` object and associated event listeners
- `onLoadComplete()` — Used to handle the situation when an audio file has completed loading
- `onID3Complete()` — Used to handle the situation when ID3 information has been returned

As discussed earlier, there are numerous ways to play a sound file in Flash Lite. So in creating the `SoundProxy` we'll demonstrate how to handle those different methods.

1. Start by creating the familiar proxy structure; ensure that you include a variable for the sound volume, called `volume`; a flag for whether the sound is muted, called `isMuted`; and the sound object itself, called `sound` (see Listing 7-36).



LISTING 7-36: Creating SoundProxy for the Media Console

```
import com.wrox.chapter7.ApplicationFacade;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter7.model.SoundProxy extends Proxy implements IProxy {

      public static var NAME:String = "SoundProxy";
```

continues

www.it-ebooks.info

LISTING 7-36 *(continued)*

```
        public var isMuted:Boolean = false;
        private var sound:Sound;
        private var volume:Number = 100;

        public function SoundProxy(data:Object) {
              super(NAME, data);
        }

        public function onRegister():Void { }

        public function loadSound(url:String):Void { }

        public function attachSound(soundContainer:MovieClip):Void { }

        public function mute():Void { }

        public function unmute():Void { }

        public function toggleSound():Void { }

        private function onLoadComplete(hasLoaded:Boolean):Void { }

        public function start():Void { }

        private function onID3Complete():Void { }

}
```

2. Next, create the `onRegister()` method. Here, you instantiate the sound object and assign handlers for the `onLoad` and `onID3` events (see Listing 7-37). We'll return to these handlers shortly.



LISTING 7-37: The onRegister method for SoundProxy

```
public function onRegister():Void {
      sound = new Sound();
      sound.onLoad = Delegate.create(this, onLoadComplete);
      sound.onID3 = Delegate.create(this, onID3Complete);
}
```

3. Next, define the `start()` method. This method should take in two parameters. The first parameter should be the number of seconds into the track `secondsOffset()`, and the second parameter should be the number of times the sound `loops` (see Listing 7-38).



LISTING 7-38: The start method for SoundProxy

```
public function start(secondsOffset:Number, loops:Number):Void {
      sound.start(secondsOffset, loops);
}
```

www.it-ebooks.info

4. For the `onLoadComplete()` method, you need to ensure that the media has loaded by using the `hasLoaded` parameter. Then, call the `start()` method and specify 0 for both `secondsOffset` and number of `loops` (see Listing 7-39).



LISTING 7-39: The onLoadComplete method for SoundProxy

```
private function onLoadComplete(hasLoaded:Boolean):Void {
      if (hasLoaded) start(0, 0);
}
```

5. Create the handler for `onID3()`. Here, you're just tracing the data output, which you could expand the application to include (see Listing 7-40).



LISTING 7-40: The onID3Complete method for SoundProxy

```
private function onID3Complete(hasID3:Boolean):Void {
      if(hasID3){
            for (var propertyName in sound.id3) {
                  trace(propertyName + " = " + sound.id3[propertyName]);
            }
      } else {
            trace("no ID3 data");
      }
}
```

6. Create the `loadSound()` method (see Listing 7-41).



LISTING 7-41: The loadSound method SoundProxy

```
public function loadSound(url:String):Void {
      sound.loadSound(url, false);
}
```

7. Create the `attachSound()` method. The method should take a MovieClip parameter, called `soundContainer`, which then needs to be passed as a parameter to the constructor of the `Sound` object (see Listing 7-42).



LISTING 7-42: The attachSound method for SoundProxy

```
public function attachSound(soundContainer:MovieClip):Void {
      sound = new Sound(soundContainer);
      sound.onID3 = Delegate.create(this, onID3Complete);
}
```

www.it-ebooks.info

8. Finally, create the methods for controlling the volume in media console: `mute()`, `unmute()`, and `toggleSound()`. For both the `unmute()` and `mute()` methods, you need to set the `isMuted` flag and call the `setVolume()` method on the sound object. In the `mute()` method, you need to save a reference to the current `volume` level of the sound object, using the `getVolume()` method, and then pass 0 as the parameter for the `setVolume()` call; the `isMuted` flag is set to `true`. In the `unmute()` method, you pass the `volume` variable to the `setVolume()` call and simply set the `isMuted` flag to `false`. The `toggleSound()` method simply calls either method, depending on the `isMuted` flag (see Listing 7-43).



LISTING 7-43: The mute, unmute, and toggleSound methods for SoundProxy

```
public function mute():Void {
      volume = sound.getVolume();
      sound.setVolume(0);
      isMuted = true;
}

public function unmute():Void {
      sound.setVolume(volume);
      isMuted = false;
}

public function toggleSound():Void {
      if(isMuted) {
            unmute();
      } else {
            mute();
      }
}
```

Having completed the `SoundProxy`, the model section of the application is now complete.

Let's just recap what you've accomplished in this section:

1. First, you created the `PlaylistProxy` and determined how the application should set and retrieve playlist items for playback of media files on a server.
2. You then created `MediaProxy` and determined how to use the net connection and net stream classes to playback.
3. Finally, you created a `SoundProxy` and determined how to start audio files and control volume.

Our attention now turns to the controller.

Creating the Media Console's Controller

In this section you'll examine each of the new commands created for the Media Console (see Figure 7-3).

www.it-ebooks.info

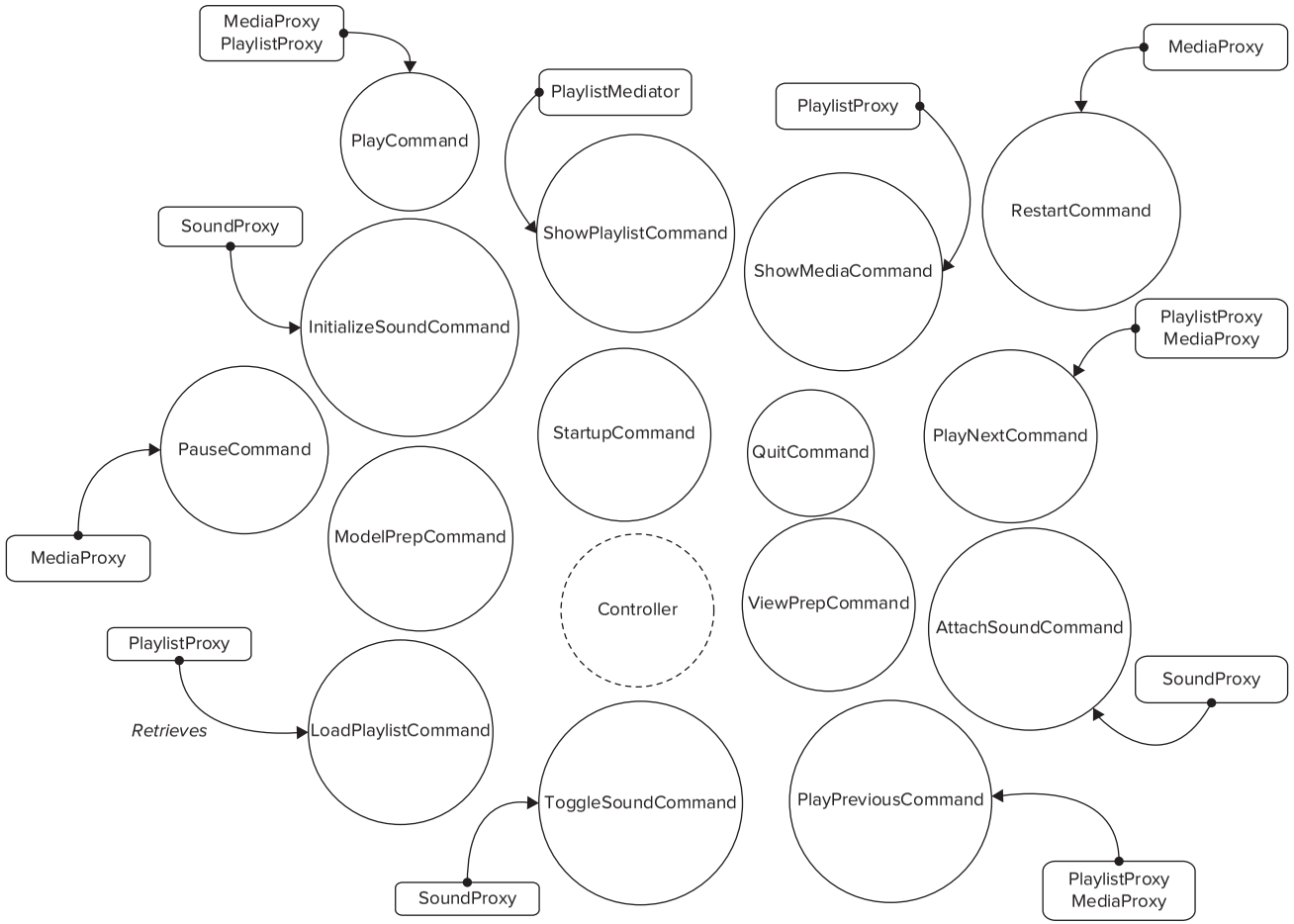


FIGURE 7-3

www.it-ebooks.info

In this section you'll cover the commands responsible for:

- Preparing the application
- Retrieving the playlist
- Controlling the media
- Manipulating the sound

The Project panel, shown in Figure 7-4, lists each of the files under the controller directory.



FIGURE 7-4

Preparing the Application

As in the previous chapter, the `StartupCommand`, `ModelPrepCommand`, and `ViewPrepCommand` are responsible for the preparation of the Media Console.

StartupCommand.as

Remember that `StartupCommand` is actually a custom class that extends `MacroCommand`. Here, you override the inherited `initializeMacroCommand()` to sequentially execute the custom `ModelPrepCommand` and `ViewPrepCommand` (see Listing 7-44).

Available for download on Wrox.com

LISTING 7-44: StartupCommand for Media Console

```
import com.wrox.chapter7.controller.ModelPrepCommand;
import com.wrox.chapter7.controller.ViewPrepCommand;
import org.puremvc.as2.patterns.command.MacroCommand;

class com.wrox.chapter7.controller.StartupCommand extends MacroCommand {

      public function initializeMacroCommand():Void {
            addSubCommand(new ModelPrepCommand());
            addSubCommand(new ViewPrepCommand());
      }
}
```

On the first frame of the timeline in `chapter7_MediaConsole.fla`, the `startup()` method is called, via the `ApplicationFacade` instance called `app`. The `STARTUP` notification, mapped to the `StartupCommand`, then calls the `initializeMacroCommand()`.

www.it-ebooks.info

ModelPrepCommand.as

The `ModelPrepCommand` registers the four proxy classes used in the application, as mentioned earlier: `FSCommandsProxy`, `PlaylistProxy`, `MediaProxy`, and `SoundProxy` (see Listing 7-45).



LISTING 7-45: The ModelPrepCommand for Media Console

```
import com.wrox.chapter7.model.FSCommandsProxy;
import com.wrox.chapter7.model.PlaylistProxy;
import com.wrox.chapter7.model.MediaProxy;
import com.wrox.chapter7.model.SoundProxy;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.command.SimpleCommand;
import org.puremvc.as2.patterns.observer.Notification;

class com.wrox.chapter7.controller.ModelPrepCommand extends SimpleCommand {

      public function execute(note:INotification):Void {
            facade.registerProxy(new FSCommandsProxy());
            facade.registerProxy(new PlaylistProxy());
            facade.registerProxy(new MediaProxy());
            facade.registerProxy(new SoundProxy());
      }

}
```

ViewPrepCommand.as

`ViewPrepCommand` registers four mediators for the application: `StatusMediator`, `PlaylistMediator`, `MediaMediator`, and `SoftKeyMediator` (see Listing 7-46) .



LISTING 7-46: The ViewPrepCommand for Media Console

```
import com.wrox.chapter7.ApplicationFacade;
import com.wrox.chapter7.view.SoftKeyMediator;
import com.wrox.chapter7.view.WaitMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.command.SimpleCommand;
import org.puremvc.as2.patterns.observer.Notification;

class com.wrox.chapter7.controller.ViewPrepCommand extends SimpleCommand {

      public function execute(note:INotification):Void {
            var mc:MovieClip = MovieClip(note.getBody());
            facade.registerMediator(new SoftKeyMediator(mc));
            facade.registerMediator(new StatusMediator(mc));
```

continues

www.it-ebooks.info

LISTING 7-46 *(continued)*

```
            facade.registerMediator(new PlaylistMediator(mc));
            facade.registerMediator(new MediaMediator(mc));
        }

    }
```

Before taking a look at these mediators, we'll cover each of the commands used in the application.

Retrieving the Playlist

Retrieving the media for the console will simply require loading the playlist, which will be invoked via the `LoadPlaylistCommand`.

LoadPlaylistCommand.as

The `LoadPlaylistCommand` calls the `loadPlaylist()` method of `PlaylistProxy` when executed, as discussed earlier (see Listing 7-47).



LISTING 7-47: The execute method for LoadPlaylistCommand

```
public function execute(note:INotification):Void {
        var pProxy:PlaylistProxy;
        pProxy = PlaylistProxy(facade.retrieveProxy(PlaylistProxy.NAME));
        pProxy.loadPlaylist();
}
```

Controlling the Media

Here you'll learn how to control the media.

PlayCommand.as

The playback of media will be executed from the `PlayCommand`, which is invoked when the user clicks on the play button in the media controls. For `PlayCommand` you need to ensure that you have imported the `MediaProxy` and the `PlaylistProxy`. The `execute()` method of `PlayCommand` invokes the resume() method of the `MediaProxy` if it is paused, and the `init()` method if it isn't, passing in references to the media server, file, media type, and duration, all obtained from the `PlaylistProxy` (see Listing 7-48).



LISTING 7-48: The execute method for PlayCommand

```
public function execute( note:INotification):Void {

        var mProxy:MediaProxy;
        mProxy = MediaProxy(facade.retrieveProxy(MediaProxy.NAME));

        var pProxy:PlaylistProxy;
        pProxy = PlaylistProxy(facade.retrieveProxy(PlaylistProxy.NAME));
```

www.it-ebooks.info

```
        if (mProxy.isPaused) {
              mProxy.resume();
        } else {
              mProxy.init( pProxy.getServer(), pProxy.getFile(), pProxy.getType(),
                    pProxy.getDuration() );
        }
  }
```

PauseCommand.as

When a user clicks the pause button, the `PauseCommand` is executed, simply calling the `pause()` method of the `MediaProxy` instance (see Listing 7-49).



LISTING 7-49: The execute method for PauseCommand

```
public function execute(note:INotification):Void {
      var mProxy:MediaProxy = MediaProxy(facade.retrieveProxy(MediaProxy.NAME));
      mProxy.pause();
}
```

PlayNextCommand.as and PlayPreviousCommand.as

Both `PlayNextCommand` and `PlayPreviousCommand` use the `PlaylistProxy` to set the media item in the list (see Listing 7-50). For the `PlayPreviousCommand`, instead of the call to `setNextItem()`, you use `setPreviousItem()` (see Listing 7-51). Before either of these methods is called, the `closeConnection()` in the `MediaProxy` is invoked. Both commands invoke the `PLAYLIST_ITEM_CHANGED` notification in the `PlaylistProxy`, which will be handled by the `MediaMediator`.



LISTING 7-50: The execute method for PlayNextCommand

```
public function execute(note:INotification):Void {

      var mProxy:MediaProxy;
      mProxy = MediaProxy(facade.retrieveProxy(MediaProxy.NAME));
      mProxy.closeConnection();

      var pProxy:PlaylistProxy;
      pProxy = PlaylistProxy(facade.retrieveProxy(PlaylistProxy.NAME));
      pProxy.setNextItem();
}
```



LISTING 7-51: The execute method for PlayPreviousCommand

```
public function execute(note:INotification):Void {

      var mProxy:MediaProxy;
      mProxy = MediaProxy(facade.retrieveProxy(MediaProxy.NAME));
      mProxy.closeConnection();
```

continues

www.it-ebooks.info

LISTING 7-51 *(continued)*

```
        var pProxy:PlaylistProxy;
        pProxy = PlaylistProxy(facade.retrieveProxy(PlaylistProxy.NAME));
        pProxy.setPreviousItem();
    }
```

RestartCommand.as

For restarting media playback, the application uses the `RestartCommand`. The `RestartCommand` calls the `playFrom()` method on the `MediaProxy` (see Listing 7-52).



LISTING 7-52: The execute method for RestartCommand

```
public function execute(note:INotification):Void {
        var mProxy:MediaProxy = MediaProxy(facade.retrieveProxy(MediaProxy.NAME));
        mProxy.playFrom(0);
}
```

Manipulating the Sound

As previously discussed, the `SoundProxy` is used as a single gateway to manipulate sound in the media console.

ToggleSoundCommand.as

The `ToggleSoundCommand` handles the call to alter the volume of the sound and, essentially, toggles between mute and unmuted sound (see Listing 7-53).



LISTING 7-53: The execute method for ToggleSoundCommand

```
public function execute(note:INotification):Void {
        var sProxy:SoundProxy = SoundProxy(facade.retrieveProxy(SoundProxy.NAME));
        sProxy.toggleSound();
}
```

Having gone through each of the commands for the controller, next take a look at the mediators used in the Media Console.

Exploring the View's Mediators

In this section you'll examine each of the new mediators created for the Media Console. The view consists of four core mediators: `PlaylistMediator`, `MediaMediator`, `StatusMediator`, and `SoftKeysMediator`. The `PlaylistMediator` uses the `ListDualRowWithIcon` component, covered in Chapter 4, to display artist and track names in the application, while the `MediaMediator` uses a custom component, consisting of several buttons to control media playback, contained in a `MediaControl` class, and a progress bar component to show the duration of the media and playback time, called `MediaProgress`. The `StatusMediator` consists of the `StatusBar` component and `Title` components, both covered in Chapter 4, while the `SoftKeysMediator` contains the `SoftKeys` component, also covered in Chapter 4 (see Figure 7-5).

www.it-ebooks.info

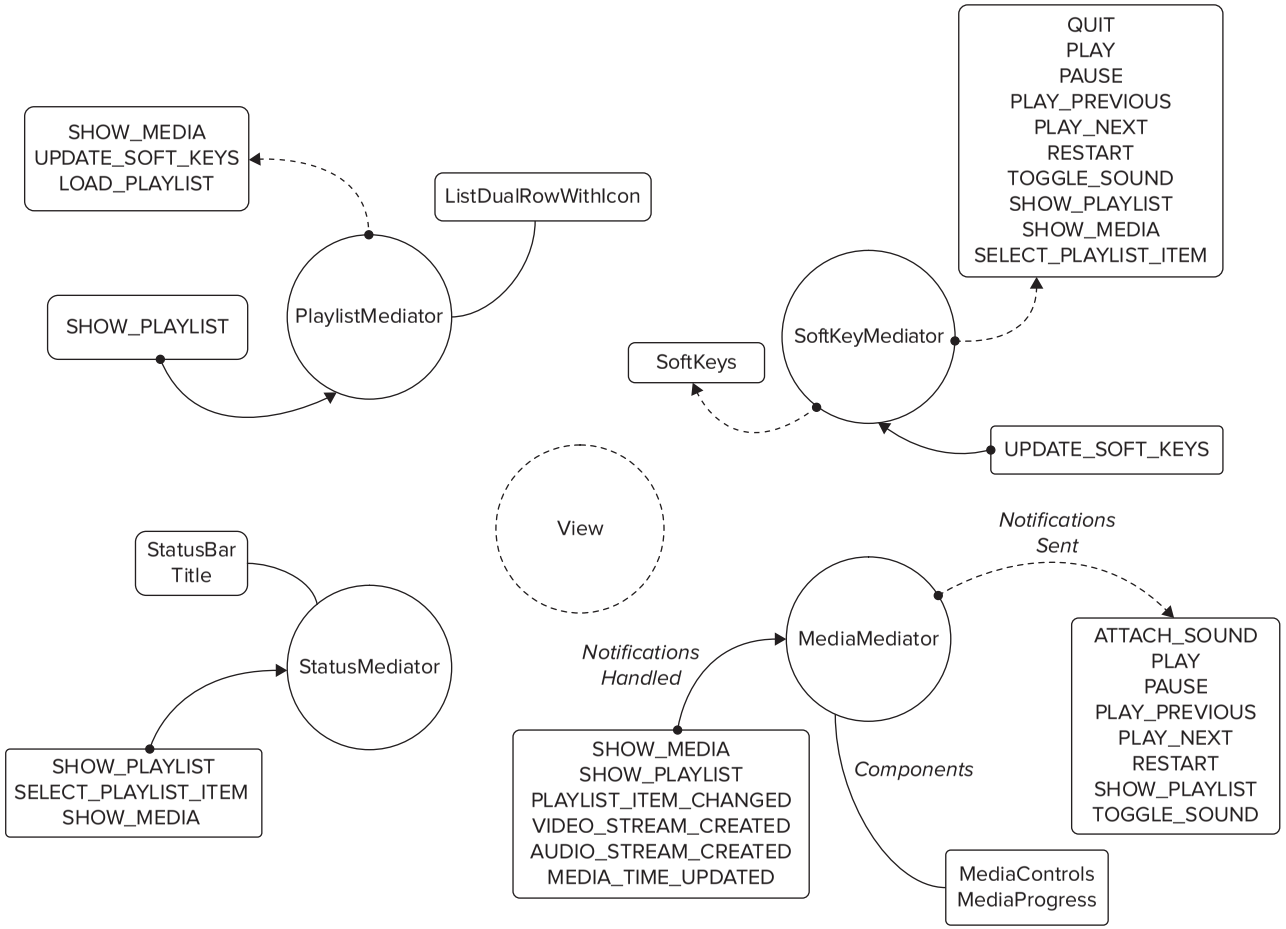


FIGURE 7-5

www.it-ebooks.info

The Project panel, shown in Figure 7-6, lists each of the files under the view directory.



FIGURE 7-6

Each mediator is registered in the `ViewPrepCommand`, which was covered earlier.

Displaying Media Playlist

Now you'll display the media playlist.

PlaylistMediator.as

Follow these steps:

1. First create the skeleton for the `PlaylistMediator`. The playlist component will be retrieved from the library. Include the `playList` and `hasPlaylistLoaded` variables (see Listing 7-54).



LISTING 7-54: Creating the PlaylistMediator class

```
import com.wrox.chapter7.ApplicationFacade;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter7.view.PlaylistMediator extends Mediator implements IMediator {
```

www.it-ebooks.info

```
        public static var NAME:String = "PlaylistMediator";

        private var playList:MovieClip;
        private var hasPlaylistLoaded:Boolean;

        public function PlaylistMediator(viewComponent:Object) {
              super(NAME, viewComponent);
        }

        public function get _component():MovieClip {
              return MovieClip(viewComponent);
        }
}
```

2. Next, create the `onRegister()` method. Here, you should send the `LOAD_PLAYLIST` notification name to retrieve data for the media console; the `hasPlaylistLoaded` variable should be set to `false` (see Listing 7-55).



LISTING 7-55: The onRegister method for PlaylistMediator

```
public function onRegister():Void {
      var depth:Number = _component.getNextHighestDepth();
      playList = _component.attachMovie("ListTwoRowWithIcon", "playList", depth);
      playList.onSelect = Delegate.create(this, onSelectHandler);
      playList._backgroundColor = 0xBFC7CD;
      playList._highlightColor = 0x526069;
      playList._scrollSliderColor = 0x526069;
      playList._x = Stage.width;
      playList._y = 58;

      hasPlaylistLoaded = false;
      sendNotification(ApplicationFacade.LOAD_PLAYLIST);
}
```

3. Next add the `SHOW_PLAYLIST` notification to the list of notifications that the mediator is interested in; also create the stub for the `handleNotification()` method (see Listing 7-56).



LISTING 7-56: The listNotification and handleNotification methods for PlaylistMediator

```
public function listNotificationInterests():Array {
      return [ApplicationFacade.SHOW_PLAYLIST];
}

public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_PLAYLIST:
            break;
      }
}
```

www.it-ebooks.info

4. When the `SHOW_PLAYLIST` notification is sent, the `PlaylistMediator` should assign the `titleArr` and `artistArr` values to the playlist component, but only when the playlist hasn't loaded. The alignment of the list is also set here, along with the assignment of the `onSelectHandler()` (see Listing 7-57).



LISTING 7-57: The handleNotification method for PlaylistMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_PLAYLIST:
                  if(!hasPlaylistLoaded){
                        playList._text1 = note.getBody().titleArr;
                        playList._text2 = note.getBody().artistArr;
                        hasPlaylistLoaded = true;
                  }

                  playList._x = 0;
                  playList.onSelect = Delegate.create(this, onSelectHandler);
            break;
      }
}
```

5. For the `onSelectHandler()` method, you need to retrieve the current item in focus. The value returned from `getFocusIndex()` needs to be sent in the notification body of the `SELECT_PLAYLIST_ITEM` (see Listing 7-58).



LISTING 7-58: The onSelectHandler method for PlaylistMediator

```
public function onSelectHandler():Void {
      switch(Key.getCode()) {
            case Key.ENTER:
                  var index:Number = playList.getFocusIndex();
                  if (index == "undefined") index = 0;

                  playList._isEnabled = false;
                  playList._x = 0-Stage.width;
                  sendNotification(ApplicationFacade.SELECT_PLAYLIST_ITEM, index);
            break;
      }
}
```

6. Finally, create the `updateSoftKeys()` method so that when the left soft key is pressed the application quits. This will be handled by the `SoftKeysMediator`. This needs to be called when the `SHOW_PLAYLIST` notification is handled by the mediator (see Listing 7-59).

www.it-ebooks.info



LISTING 7-59: The updateSoftKeys method for PlaylistMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_PLAYLIST:
                  if(!hasPlaylistLoaded){
                        playList._text1 = note.getBody().titleArr;
                        playList._text2 = note.getBody().artistArr;
                        hasPlaylistLoaded = true;
                  }

                  playList._x = 0;
                  playList.onSelect = Delegate.create(this, onSelectHandler);
                  updateSoftKeys();
            break;
      }
}

private function updateSoftKeys():Void {
      var msk:Object = {label:" Select ", note:""};
      var lsk:Object = {label:" Quit ", note:ApplicationFacade.QUIT};
      var rsk:Object = {label:"", note:""};
      sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
}
```

Displaying the Media Components

Next, you'll display the media components.

MediaMediator.as

This mediator is responsible for displaying the currently selected item in the playlist and user's input, controlling media playback of audio and video. It contains two core components:

- `MediaProgress`
- `MediaControls`

It should also retrieve an audio symbol from the library of the `.fla`, and draw two text fields to display the artist and title of the playlist item. In the following steps you create the `MediaMediator` class.

1. Declare private variables for the core elements of `MediaMediator`: `video`, `controls`, `progress`, `artistTxt`, `titleTxt`, `audioSymbol`, `videoContainer`, and `soundContainer` (see Listing 7-60).



LISTING 7-60: Defining private variables for MediaMediator

```
import com.wrox.chapter7.ApplicationFacade;
import com.wrox.chapter7.view.components.MediaControls;
import com.wrox.chapter7.view.components.MediaProgress;
```

continues

www.it-ebooks.info

LISTING 7-60 *(continued)*

```
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter7.view.MediaMediator extends Mediator implements IMediator {

      public static var NAME:String = "MediaMediator";

      private var video:Video;

      private var controls:MediaControls;
      private var progress:MediaProgress;

      private var artistTxt:TextField;
      private var titleTxt:TextField;

      private var audioSymbol:MovieClip;
      private var videoContainer:MovieClip;
      private var soundContainer:MovieClip;

      public function MediaMediator(viewComponent:Object) {
            super(NAME, viewComponent);
      }

}
```

2. Next, override the `onRegister()` method for the mediator. Here, you need to attach and position the `audioSymbol` from the library of the `.fla` file, this also needs to be hidden here. The `videoObject` and `soundObject` also need to be attached and assigned to the variables `videoContainer` and `soundContainer`, respectively. The controls and progress components should be instantiated, along with creating the artist and title text fields. Notice that the depth for each asset is retrieved from the `_component`, so one isn't overwritten with the other (see Listing 7-61).



LISTING 7-61: The onRegister method for MediaMediator

```
public function onRegister():Void {

      var aDepth:Number = _component.getNextHighestDepth();
      audioSymbol = _component.attachMovie("audioSymbol", "audioSymbol", aDepth);
      audioSymbol._x = Math.round( (Stage.width/2)-(audioSymbol._width/2) );
      audioSymbol._y = 100;
      audioSymbol._visible = false;

      var vDepth:Number = _component.getNextHighestDepth();
      videoContainer = _component.attachMovie("videoObject", "videoObj", vDepth);
      video = videoContainer.video;
```

www.it-ebooks.info

```
        var sDepth:Number = _component.getNextHighestDepth();
        soundContainer = _component.attachMovie("soundObject", "soundObj", sDepth);

        controls = new MediaControls(_component);
        progress = new MediaProgress(_component);

        video = videoContainer.video;

        var textFormat:TextFormat = new TextFormat();
        textFormat.font = "Arial";
        textFormat.size = 12;
        textFormat.color = 0xFFFFFF;
        textFormat.align = "center";

        var t1Depth:Number = _component.getNextHighestDepth();
        _component.createTextField("artistTxt", t1Depth, 0, 70, 240, 40);
        artistTxt = new TextField();
        artistTxt = _component.artistTxt;
        artistTxt.setNewTextFormat(textFormat);

        var t2Depth:Number = _component.getNextHighestDepth();
        _component.createTextField("titleTxt", t2Depth, 0, 170, 240, 40);
        titleTxt = new TextField();
        titleTxt = _component.titleTxt;
        titleTxt.setNewTextFormat(textFormat);
        titleTxt.text = "";

}
```

3. Add each of the notifications the `MediaMediator` will be interested in via the `listNotificationInterests()` method. The list should include `SHOW_MEDIA`, `SHOW_PLAYLIST`, `PLAYLIST_ITEM_CHANGED`, `AUDIO_STREAM_CREATED`, `VIDEO_STREAM_CREATED`, and `MEDIA_TIME_UPDATED` (see listing 7-62).



LISTING 7-62: The listNotificationInterests method for MediaMediator

```
public function listNotificationInterests():Array {
        return [ ApplicationFacade.SHOW_MEDIA,
                 ApplicationFacade.SHOW_PLAYLIST,
                 ApplicationFacade.PLAYLIST_ITEM_CHANGED,
                 ApplicationFacade.AUDIO_STREAM_CREATED,
                 ApplicationFacade.VIDEO_STREAM_CREATED,
                 ApplicationFacade.MEDIA_TIME_UPDATED ];
}
```

4. Following on from the notification interests, define the stubs for the `handleNotification()` method, including each of the notification names mentioned in the previous step (see Listing 7-63).

www.it-ebooks.info



LISTING 7-63: The handleNotification method for MediaMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_MEDIA:
            break;
            case ApplicationFacade.SHOW_PLAYLIST:
            break;
            case ApplicationFacade.PLAYLIST_ITEM_CHANGED:
            break;
            case ApplicationFacade.AUDIO_STREAM_CREATED:
            break;
            case ApplicationFacade.VIDEO_STREAM_CREATED:
            break;
            case ApplicationFacade.MEDIA_TIME_UPDATED:
            break;
      }
}
```

5. For the `SHOW_MEDIA` notification, you need to make sure that the view elements of the component are visible. Here, make calls to the `showComponents()` and `activateControls()` methods; you'll add the detail to these later. For the `SHOW_PLAYLIST` notification, make a call to the `hideComponents()` and `deactivateControls()` methods. You'll return to the `handleNotification()` method shortly (see Listing 7-64).



LISTING 7-64: Handling the SHOW_MEDIA and SHOW_PLAYLIST notifications in the handleNotification method for MediaMediator

```
public function handleNotification(note:INotification):Void {

      switch(note.getName()) {
            case ApplicationFacade.SHOW_MEDIA:
                  showComponents();
                  activateControls();
            break;
            case ApplicationFacade.SHOW_PLAYLIST:
                  hideComponents();
                  deactivateControls();
            break;
            case ApplicationFacade.PLAYLIST_ITEM_CHANGED:
            break;
            case ApplicationFacade.AUDIO_STREAM_CREATED:
            break;
            case ApplicationFacade.VIDEO_STREAM_CREATED:
            break;
            case ApplicationFacade.MEDIA_TIME_UPDATED:
            break;
      }
}
```

www.it-ebooks.info

```
public function showComponents():Void { }

public function hideComponents():Void { }

public function activateControls():Void { }

public function deactivateControls():Void { }
```

6. For the `showComponents()` method, ensure that each of the components for the mediator is aligned and visible to the screen; for the `hideComponents()` method, ensure that each component is hidden (see Listing 7-65).



LISTING 7-65: The showComponents and hideComponents methods for MediaMediator

```
public function showComponents():Void {
      controls.moveTo(0, 230);
      progress.moveTo(10, 200);
      artistTxt._x = 0
      titleTxt._x = 0;
      audioSymbol._visible = true;
}

public function hideComponents():Void {
      controls.moveTo(Stage.width, 235);
      progress.moveTo(Stage.width, controls._y -30);
      artistTxt._x = Stage.width;
      titleTxt._x = Stage.width;
      audioSymbol._visible = false;
}
```

7. Complete the `activateControls()` method. Here, you need to ensure that each of the events dispatched from the controls component is handled via an `onClick()` method (see Listing 7-66).



LISTING 7-66: The activateControls method for MediaMediator

```
public function activateControls():Void {
      controls.addEventListener("onPlay", Delegate.create(this, onClick));
      controls.addEventListener("onPause", Delegate.create(this, onClick));
      controls.addEventListener("onPrevious", Delegate.create(this, onClick));
      controls.addEventListener("onNext", Delegate.create(this, onClick));
      controls.addEventListener("onRestart", Delegate.create(this, onClick));
      controls.addEventListener("onMute", Delegate.create(this, onClick));
      controls.addEventListener("onUnmute", Delegate.create(this, onClick));
}

private function onClick():Void { }
```

8. Next, add a `switch()` statement to the `onClick()` method. Each `case` statement needs to send a notification based on the control that was selected by the user. The type property of the `event` object is used to differentiate between the calls (see Listing 7-67).

www.it-ebooks.info



LISTING 7-67: The onClick method for MediaMediator

```
public function onClick(event:Object):Void {
      switch(event.type){
            case "onPlay":
                  sendNotification(ApplicationFacade.PLAY);
            break;
            case "onPause":
                  sendNotification(ApplicationFacade.PAUSE);
            break;
            case "onPrevious":
                  sendNotification(ApplicationFacade.PLAY_PREVIOUS);
            break;
            case "onNext":
                  sendNotification(ApplicationFacade.PLAY_NEXT);
            break;
            case "onRestart":
                  sendNotification(ApplicationFacade.RESTART);
            break;
            case "onMute":
                  sendNotification(ApplicationFacade.TOGGLE_SOUND);
            break;
            case "onUnmute":
                  sendNotification(ApplicationFacade.TOGGLE_SOUND);
            break;
      }
}
```

9. For the `deactivateControls()` method, the aim here is to remove the event listeners from the controls (see Listing 7-68).



LISTING 7-68: The execute method for PlayCommand

```
public function deactivateControls():Void {
      controls.removeEventListener("onPlay");
      controls.removeEventListener("onPause");
      controls.removeEventListener("onPrevious");
      controls.removeEventListener("onNext");
      controls.removeEventListener("onRestart");
      controls.removeEventListener("onMute");
      controls.removeEventListener("onUnmute");
}
```

10. Returning to the `handleNotification()` method, the `video.clear()` method is called in preparation for the change in playlist item, and once the artist and title values are assigned to their corresponding text fields, the `PLAY` notification can be sent (see Listing 7-69).

www.it-ebooks.info



LISTING 7-69: Handling the PLAYLIST_ITEM_CHANGED notification in handleNotification for MediaMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_MEDIA:
                  showComponents();
                  activateControls();
            break;
            case ApplicationFacade.SHOW_PLAYLIST:
                  hideComponents();
                  deactivateControls();
            break;
            case ApplicationFacade.PLAYLIST_ITEM_CHANGED:
                  video.clear();
                  artistTxt.text = note.getBody().artist;
                  titleTxt.text = note.getBody().title;
                  sendNotification(ApplicationFacade.PLAY);
            break;
            case ApplicationFacade.AUDIO_STREAM_CREATED:
            break;
            case ApplicationFacade.VIDEO_STREAM_CREATED:
            break;
            case ApplicationFacade.MEDIA_TIME_UPDATE:
            break;
      }
}
```

11. Complete the cases for `AUDIO_STREAM_CREATED` and `VIDEO_STREAM_CREATED`. Here, the notification body sent from the `MediaProxy` should contain the `NetStream` instance, so this needs to be attached to the `soundContainer` MovieClip, and then subsequently sent as the notification body of `ATTACH_SOUND` (see Listing 7-70).



LISTING 7-70: Handling the AUDIO_STREAM_CREATED and VIDEO_STREAM_CREATED notifications in the handleNotification method for MediaMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_MEDIA:
                  showComponents();
                  activateControls();
            break;
            case ApplicationFacade.SHOW_PLAYLIST:
                  hideComponents();
                  deactivateControls();
            break;
            case ApplicationFacade.PLAYLIST_ITEM_CHANGED:
                  video.clear();
                  artistTxt.text = note.getBody().artist;
                  titleTxt.text = note.getBody().title;
                  sendNotification(ApplicationFacade.PLAY);
            break;
```

continues

www.it-ebooks.info

LISTING 7-70 *(continued)*

```
            case ApplicationFacade.AUDIO_STREAM_CREATED:
                  soundContainer.attachAudio(note.getBody());
                  sendNotification(ApplicationFacade.ATTACH_SOUND, soundContainer);
            break;
            case ApplicationFacade.VIDEO_STREAM_CREATED:
                  video.attachVideo(note.getBody());
                  soundContainer.attachAudio(note.getBody());
                  sendNotification(ApplicationFacade.ATTACH_SOUND, soundContainer);
            break;
            case ApplicationFacade.MEDIA_TIME_UPDATED:
            break;
      }
}
```

12. The last notification that needs to be handled in the `handleNotification()` is the `MEDIA_TIME_UPDATED` notification. This notification is sent from the `MediaProxy`, in the `checkTime()` method, and sends the current and duration time of the `NetStream`. These values should be passed on to the `updateProgess()` method of the progress component (see Listing 7-71).



LISTING 7-71: Handling the MEDIA_TIME_UPDATE notification in the handleNotification method for MediaMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_MEDIA:
                  showComponents();
                  activateControls();
            break;
            case ApplicationFacade.SHOW_PLAYLIST:
                  hideComponents();
                  deactivateControls();
            break;
            case ApplicationFacade.PLAYLIST_ITEM_CHANGED:
                  video.clear();
                  sendNotification(ApplicationFacade.PLAY);
            break;
            case ApplicationFacade.AUDIO_STREAM_CREATED:
                  soundContainer.attachAudio(note.getBody());
                  sendNotification(ApplicationFacade.ATTACH_SOUND, soundContainer);
            break;
            case ApplicationFacade.VIDEO_STREAM_CREATED:
                  video.attachVideo(note.getBody());
                  soundContainer.attachAudio(note.getBody());
                  sendNotification(ApplicationFacade.ATTACH_SOUND, soundContainer);
            break;
            case ApplicationFacade.MEDIA_TIME_UPDATED:
                  var currentTime:Number = note.getBody()[0];
                  var durationTime:Number = note.getBody()[1];
```

www.it-ebooks.info

```
                progress.updateProgress(currentTime, durationTime);
            break;
        }
    }
```

StatusMediator.as

Lastly, the `StatusMediator` class contains the decorative components for the `Title` and `StatusBar`. It simply captures three notifications and displays appropriate text in the `Title` component accordingly. The full class is shown in Listing 7-72.



LISTING 7-72: The StatusMediator class

```
import com.wrox.chapter7.ApplicationFacade;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter7.view.StatusMediator extends Mediator implements IMediator {

      public static var NAME:String = "StatusMediator";

      private var statusBar:MovieClip;
      private var title:MovieClip;

      public function StatusMediator(viewComponent:Object) {
            super(NAME, viewComponent);
      }

      public function getMediatorName():String {
            return StatusMediator.NAME;
      }

      public function onRegister():Void {
            var sDepth:Number = _component.getNextHighestDepth();
            statusBar = _component.attachMovie("StatusBar", "statusBar", sDepth);
            var tDepth:Number = _component.getNextHighestDepth();
            title = _component.attachMovie("Title", "title", tDepth);
      }

      public function listNotificationInterests():Array {
            return [ ApplicationFacade.SHOW_PLAYLIST,
                     ApplicationFacade.SELECT_PLAYLIST_ITEM,
                     ApplicationFacade.SHOW_MEDIA ];
      }

      public function handleNotification(note:INotification):Void {
            switch (note.getName()){

                  case ApplicationFacade.SHOW_PLAYLIST:
                        title._text = "Playlist";
                  break;
                  case ApplicationFacade.SELECT_PLAYLIST_ITEM:
                        title._text = "Now playing";
                  break;
```

continues

www.it-ebooks.info

LISTING 7-72 *(continued)*

```
                case ApplicationFacade.SHOW_MEDIA:
                      title._text = "Now playing";
                break;
          }
    }

    public function get _component():MovieClip {
          return MovieClip(viewComponent);
    }
}
```

Viewing the Application in Device Central

You have now covered each aspect of the Media Console. If you haven't done so already, run the completed app in Device Central, and ensure that you have selected the Nokia N95 8GB for testing. Figure 7-7 shows the playlist view for the application with the second track highlighted.

When you select the track, you need to navigate to the play button and select it. During playback of the track, you will see the progress bar updated and the time elapse for the selected item, along with its duration.

You can also mute the volume of the media, restart the track, and navigate to the next items in the playlist. The title and artist names are also shown, and updated when the selected playlist item changes (see Figure 7-8).



FIGURE 7-7



FIGURE 7-8

www.it-ebooks.info

SUMMARY

In this chapter, you've learned how to use audio and video in your Flash Lite applications, using `NetConnection` and `NetStatus` as the core classes. You also learned how to stream media files from Flash Media server.

In the next chapter, you'll take a closer look at using images in the build of a Flash Lite Image Viewer.

www.it-ebooks.info

8

Creating an Image Viewer Client

WHAT'S IN THIS CHAPTER?

- Overcoming the challenges of using images in Flash Lite
- Creating core components for Image Viewer
- Using PureMVC to build Image Viewer

It's hard to imagine, but there was a time when mobile devices weren't able to display color images. With most new devices being shipped with built-in cameras and featuring picture messaging, these days the ability to capture and/or manipulate images on a mobile device is a fairly common feature.

In this chapter, you will develop a simple Image Viewer client, a Flash mobile application that displays an array of images located on the device. We'll be referencing images from the Formula One team Brawn GP (now Mercedes GP), taken from various Grand Prix's and Practice sessions during the F1 season 2009.

THE CHALLENGES

Using images in Flash Lite has evolved greatly from the early versions of the runtime, when the only way to include images in an application was to embed them in the actual application or wrap them in a SWF container and load them dynamically into the application. Fortunately, these workarounds are no longer required.

Before we take a look at the code for the Image Viewer, in this section you'll get an idea of some of the issues you face in using images in Flash Lite applications:

- Supported images
- Loading multiple images

www.it-ebooks.info

- Memory considerations
- Handling image sizes

Supported Images

When targeting a small number of mobile devices on which to run your Flash Lite content, you can ensure that the devices support the display of the image type you want to use by detecting MIME types.

Image MIME Types

Multimedia Internet Message Extension (MIME) types are essentially standards for describing various types of information.

Here are some widely used image MIME types you may be familiar with:

- `Image/gif`
- `Image/jpeg`
- `Image/png`
- `Image/svg+xml`
- `Image/tiff`

In Chapter 3 you examined `System.capabilities` and learned that specific device properties can be retrieved from the device. The `System.capabilities.imageMIMETypes` property returns image MIME types supported by the target device. The property contains an array of supported image MIME types.

MIME types are used to indicate the type of image supported by a device. For Flash Lite applications, only GIF, JPEG, and PNG image MIME types are supported. Animated GIFs and progressive JPEGs are not supported.

If you are testing and targeting a specific device when developing Flash Lite content, very likely you will want to know what MIME types the device supports. Calling `System.capabilities.imageMIMETypes` can help you to decide on a development route for the application. For example, if you determine that a device doesn't support PNGs, you could use JPEGs or GIFs instead.

Loading Multiple Images

Because it's likely that you'll need to load multiple images into your Flash Lite content, it's important to note that there are limitations on the number of images that can be loaded into the Flash Lite Player at any one time. You must take this into consideration when creating dynamic image applications.

When displaying images, the Flash Lite Player decompresses image files into bitmaps; a 30KB image could in fact use 1MB of RAM. With the majority of devices shipping with Flash Lite 3 pre-installed and at least 8MB of RAM, you can see that there's not much freedom in terms of loading

www.it-ebooks.info

multiple images. Devices such as the Nokia 5800 Xpress Music and N95 provide 16–25MB of RAM, thereby allowing a greater number of images to be loaded.

In short, an important aspect of loading multiple images into Flash Lite is that each device by default has a limit on the number of images that can be loaded at once. This number is usually five, but it depends on the device. You therefore need to ensure that you can keep track of the number of image requests. In the section "*Exploring the ImageGrid Component*," you will see how this is achieved.

Memory Considerations

In Chapter 2 you learned about Flash Lite's limitations in terms of mobile devices' memory capacities.

It's important to remember that you can't just load an unlimited number of images into your Flash Lite application. The size of the image loaded into an application will affect memory consumption, so you need to consider whether your application will have enough memory available for loading images. Part of managing images in Flash Lite is unloading them from an application when they are no longer needed. Otherwise, memory builds up and eventually you will see "Out of Memory" error messages.

In Device Central CS4, if you run the Image Viewer application, `chapter8_imageViewer.swf`, and then open the Memory panel, you'll see that the memory consumption fluctuates while you use the application. Figure 8-1 displays the memory profile of the Image Viewer when an image has been loaded and unloaded from the view.

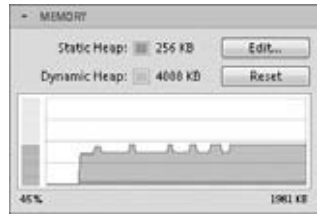


FIGURE 8-1

Handling Image Sizes

It's a good idea to use only the image sizes you need to use when creating mobile applications. For the full Flash Player, developers tend to use a variety of methods to render and resize images of various sizes. Because you are developing for the small screen, your image sizes will likely be a lot smaller. If you need to display an image in your application that is, say, 90 × 90 px on screen, then create the image file at the same dimensions prior to loading it into the application. Your overall aim here is to reduce the amount of processing power required to render images to screen, and relying on the device to resize or render the image is unnecessarily CPU-intensive. Using correct image dimensions where possible is good practice, and even applications that need to display more than one version of an image (say, a full-size version and a thumbnail) tend to have a small version and large version of the image. This approach is used in the Image Viewer.

COMPONENTS USED IN THE IMAGE VIEWER

In this section you'll take a look at the two core classes that will be used as part of the Image Viewer client:

- `MovieClipLoader`
- `SharedObject`

www.it-ebooks.info

MovieClipLoader

In addition to the `loadMovie()` method, the `MovieClipLoader` class supplies several methods that provide a way to find information about the initiated image load. You can use the methods and events of `MovieClipLoader` to good effect in your applications. The `loadClip()` method, which will be covered shortly, takes two parameters: a path to an image and the target `MovieClip` reference, which will contain your image.

Later you'll take a look at creating an image grid that needs to load several images in one view. Loading images sequentially makes it a lot easier to maintain the progress at which each image in a collection is loaded. You can track the loading progress of an image using the `getProgress()` method. This method returns the current number of bytes for the file that is being loaded in the Flash Lite Player.

The `MovieClipLoader` object essentially has five events that can be invoked:

- `onLoadComplete` — when the file is completely downloaded
- `onLoadError` — when the file fails to load
- `onLoadStart` — when the file being loaded starts
- `onLoadInit` — when the first frame of the loaded clip is executed; here you can initiate properties of the image file, such as altering the `width` or `height`
- `onProgress` — when the file is written to disk; this occurs between `onLoadStart()` and `onLoadComplete()`

Each event that is mentioned here returns a reference to the target `MovieClip`. In addition, the event handler for `onLoadComplete` will contain a reference to the listener object. The event handler for `onLoadProgress` will contain the `loadedBytes` and the `totalBytes` of the file being loaded. Each of these will be covered in the next section.

Creating an Image Component

In the following steps you'll create a custom `Image` class for loading images. The `Image` class will extend the `MovieClipLoader` class and will inherit the main functions that are used in the Image Viewer. These functions will allow you to know when the image has started loading, monitor its progress, and know when it has been added to the application.

As with the other examples in the book, you can follow the code here and create the `Image` class, and you can also download the full code example. The `Image` class will be used in this chapter and others later in the book.

1. In constructing the `Image` class you need to extend the `MovieClipLoader` and call the `super()` method in the constructor. Ensure that the `Delegate` and `EventDispatcher` are imported into the class (see Listing 8-1).

www.it-ebooks.info



LISTING 8-1: Extending MovieClipLoader for the Image class

```
import mx.utils.Delegate;
import mx.events.EventDispatcher;

class com.wrox.chapter8.view.components.Image extends MovieClipLoader {

      public var dispatchEvent:Function;
      public var addEventListener:Function;
      public var removeEventListener:Function;

      public function Image(){
            super();
            EventDispatcher.initialize(this);
      }
}
```

2. Declare the private variables for the `Image` class. In total there should be six variables to add here: A `String` reference to an image path will be saved in the variable called `_dataProvider`; then two `MovieClip` variables, one to hold an image called `_imageContainer` and the other to contain a reference to a frame decoration around the image called `_frameContainer`; a listener object for the events of the image load called `_loadListenerObj`; then finally two image settings, `_width` and `_height` (see Listing 8-2).



LISTING 8-2: Declaring private variables for the Image class

```
import mx.utils.Delegate;
import mx.events.EventDispatcher;

class com.wrox.chapter8.view.components.Image extends MovieClipLoader {

      private var _dataProvider:String;
      private var _loadListenerObj:Object;
      private var _imageContainer:MovieClip;
      private var _frameContainer:MovieClip;
      private var _width:Number;
      private var _height:Number;

      public var dispatchEvent:Function;
      public var addEventListener:Function;
      public var removeEventListener:Function;

      public function Image(){
            super();
            EventDispatcher.initialize(this);
      }
}
```

3. In the constructor for the class, add three parameters: `target`, which is a `MovieClip`; and two parameters: `width` and `height`. The `Image` class needs to know the target `MovieClip`,

www.it-ebooks.info

which will contain the image, so the `target` parameter will need to be supplied when the class is instantiated. The `width` and `height` parameters simply specify the width and height of the image when it is rendered; these values are assigned to the private `_width` and `_height` variables.

Here you also need to create an empty `MovieClip` reference for containing the image, using the `createEmptyMovieClip()` method. The method requires two arguments: The first is a unique name for the `MovieClip`, and the second is the depth for the `MovieClip`. The variable called `imageDepth` is declared here simply to retrieve the next-highest depth of `target` via `getNextHighestDepth()`. This value is then supplied as the second parameter to `createEmptyMovieClip()`. This is actually an important aspect of the `Image` class: The next depth of the target must be received so that the current depth of the target isn't overwritten. Likewise, the name value needs to be uniquely specified, so the first parameter for `createEmptyMovieClip()` consists of combining "img" with the `imageDepth` value. The new `MovieClip` created by the method should then be assigned to the `imageContainer` property (see Listing 8-3).



LISTING 8-3: Adding parameters to the constructor of the Image class

```
public function Image(target:MovieClip, width:Number, height:Number) {

        super();
        EventDispatcher.initialize(this);

        var imageDepth:Number = target.getNextHighestDepth();

        _imageContainer = target.createEmptyMovieClip("img" + imageDepth, imageDepth);

        _width = width;
        _height = height;

}
```

4. Define the `dataProvider()` method. The image path for the image called `imagePath` will be set via the `dataProvider()` setter method, and in this method you need to initialize the `_loadListenerObj` object. Here the `onLoadInit()` and `onLoadProgress()` events need to be assigned to their custom event handler methods, `onLoadInitHandler()` and `onLoadProgressHandler()`, respectively. The `addListener()` method, inherited from `MovieClipLoader`, is called here with the `_loadListenerObj` passed as the reference. Finally, the inherited `loadClip()` method is called, with `imagePath` and `imageContainer` supplied as parameters (see Listing 8-4).



LISTING 8-4: The dataProvider() method for Image

```
public function Image(target:MovieClip, width:Number, height:Number) {

        super();
        EventDispatcher.initialize(this);
```

www.it-ebooks.info

```
        var imageDepth:Number = target.getNextHighestDepth();

        _imageContainer = target.createEmptyMovieClip("img" + imageDepth, imageDepth);

        _width = width;
        _height = height;

    }

    public function set dataProvider(imagePath:String):Void {

        _loadListenerObj = new Object();
        _loadListenerObj.onLoadInit = Delegate.create(this, onLoadInitHandler);
        _loadListenerObj.onLoadProgress = Delegate.create(this, onLoadProgressHandler);

        addListener(_loadListenerObj);
        loadClip(imagePath, _imageContainer);

    }

    private function onLoadInitHandler():Void {

    }

    private function onLoadProgressHandler():Void {

    }
```

5. For the custom `onLoadInitHandler()` method, you dispatch an event to signal that an image load is complete. Once the event has been fired through `dispatchEvent()`, use the `removeListener()` method to remove `_loadListenerObj` and prevent further events from being called to the `onLoadInitHandler()` method (see Listing 8-5).



LISTING 8-5: The onLoadInitHandler for Image

```
private function onLoadInitHandler():Void {

        dispatchEvent({target:this, type:"onLoadComplete"});
        removeListener(_loadListenerObj);

}
```

The custom `onLoadProgressHandler()` method is simply demonstrating how the `onLoadProgress()` event can be handled. Three arguments are passed by the method: `target`, the reference to the `MovieClip` that is our `imageContainer`; the `loadedBytes`; and the `totalBytes`. From these values you can calculate the percentage (see Listing 8-6).

www.it-ebooks.info



LISTING 8-6: The onLoadProgressHandler for Image

```
private function onLoadProgressHandler(target:MovieClip, loadedBytes:Number,
totalBytes:Number):Void {

      trace(Math.round((loadedBytes / totalBytes) * 100) + "%");

      if (loadedBytes == totalBytes) {
            trace("done");
      }

}
```

Returning to the `onLoadInitHandler()` method, `Image` will allow a frame to be drawn around the loaded image. The `onLoadInitHandler()` method needs to call `drawFrame()` to initialize this (see Listing 8-7).



LISTING 8-7: Defining the drawFrame() method for the Image class

```
private function onLoadInitHandler():Void {

      dispatchEvent({ target:this, type:"onLoadComplete" });
      removeListener(_listenerObject);
      drawFrame();
}

private function drawFrame():Void {

      var frameDepth:Number = _target.getNextHighestDepth());
      _frameContainer = _target.createEmptyMovieClip("imageFrame", frameDepth);
}
```

6. Create the public `setAlpha()` and `setSize()` methods. The `setAlpha()` method sets the transparency of the image once it has been loaded. The `setSize()` method allows resizing of the image once it has loaded (see Listing 8-8).



LISTING 8-8: Creating the setAlpha() and setSize() methods for the Image class

```
public function setAlpha(alpha:Number):Void {
      _imageContainer._alpha = alpha;
}

public function setSize(width:Number, height:Number):Void {
      _imageContainer._width = width;
      _imageContainer._height = height;
}
```

7. Add the public `getX()`, `getY()`, `getWidth()`, `getHeight()`, and `moveTo()` methods to the `Image` class. Both the `getX()` and `getY()` methods need to retrieve the position of the image along the *x* axis and *y* axis, respectively. The `getWidth()` and `getHeight()` methods

www.it-ebooks.info

should return the width and height of the image. These can be used by the image grid, as you will see shortly. The `moveTo()` method will allow you to position the component by providing the *x* and *y* coordinates as parameters (see Listing 8-9).



LISTING 8-9: Creating the getX(), getY(), getWidth(), getHeight(), and moveTo() methods for Image

```
public function getX():Number {
      return _imageContainer._x;
}

public function getY():Number {
      return _imageContainer._y;
}

public function getWidth():Number {
      return _imageContainer._width;
}

public function getHeight():Number {
      return _imageContainer._height;
}

public function moveTo(x:Number, y:Number):Void {
      _imageContainer._x = x;
      _imageContainer._y = y;
}
```

8. Create the `destroy()` method. This needs to call the inherited `unloadClip()` method, a step that basically removes the image loaded into the `_imageContainer` from view (see Listing 8-10).



LISTING 8-10: Using the unloadClip() method to remove an image

```
public function destroy():Void {
      unloadClip(_imageContainer);
}
```

Exploring the ImageGrid Component

Having completed the `Image` component, you can now explore the `ImageGrid` component. This class actually has numerous variables, which are used for a number of methods in the class and are covered later in this section.

The `columnCount` and `rowCount` variables are declared so that the overall grid matrix can be set. The Image Viewer uses a 3 by 2 grid — 3 columns and 2 rows. Both the `imageWidth` and `imageHeight` variables are assigned to instances of the `Image` class, which the `ImageGrid` uses. The two variables, `columnPadding` and `rowPadding`, allow for padding in the columns and rows between each image also. Each of the pre-mentioned variables is set in the constructor for `ImageGrid`. The class also imports `EventsDispatcher`, so an event can be dispatched by the component to notify you that all

www.it-ebooks.info

the images for `ImageGrid` have loaded. The `totalImages` and `loadCounter` variables allow the class to keep track of the total images loaded in `ImageGrid`. The `_selectedIndex` and `_selectedItem` variables are used to keep track of the highlighted image in `ImageGrid` (see Listing 8-11).



LISTING 8-11: The constructor and properties for ImageGrid

```
private var _dataProvider:Array;
private var _mc:MovieClip;
private var _selectedItem:Number = 0;
private var _selectedIndex:Number = 0;

private var columnCount:Number;
private var columnMax:Number;
private var columnPadding:Number;
private var frame:MovieClip;
private var imageContainer:Array;
private var imageWidth;
private var imageHeight;
private var loadCounter:Number;
private var rowCount:Number;
private var rowMax:Number;
private var rowPadding:Number;
private var totalImages:Number;

public var dispatchEvent:Function;
public var addEventListener:Function;
public var removeEventListener:Function;

public function ImageGrid(target:MovieClip, w:Number, h:Number, c:Number, r:Number,
p:Number) {

      super();

       var gridDepth:Number = target.getNextHighestDepth();
       var frameDepth:Number = target.getNextHighestDepth();

       _mc = target.createEmptyMovieClip("gridContainer" + gridDepth, gridDepth);

      frame = _mc.createEmptyMovieClip("frameContainer" + fd, fd);
      imageWidth = w;
      imageHeight = h;

      rowCount = 0;
      columnCount = 0;

      rowMax = r;
      columnMax = c;

      rowPadding = p;
      columnPadding = p;

      EventDispatcher.initialize(this);

}
```

www.it-ebooks.info

As with the `Image` component, the data for `ImageGrid` is set via the `dataProvider` setter method (see Listing 8-12).



LISTING 8-12: The dataProvider method for ImageGrid

```
public function set dataProvider(imgs:Array):Void {
      _dataProvider = imgs;
      totalImages = _dataProvider.length;
      draw();
}

private function draw():Void {
      loadCounter = -1;
      loadImages();
}
```

The `loadImages()` method loops through the `totalImages` set from the `dataProvider` array. Subsequently, this method instantiates a new image with the `width` and `height` parameters from `imageWidth` and `imageHeight`. Remember that the images are not actually loaded at this point. Each image needs to be added to the `imageContainer` array so that it can be referenced dynamically. This is done via the `getNextImage()` method (see Listing 8-13).



LISTING 8-13: The loadImages() method for ImageGrid

```
private function loadImages():Void {

      var img:Image;
      var d:Number;
      var h:Number = imageHeight;
      var w:Number = imageWidth;

      imageContainer = [];

      for(var i:Number = 0; i < totalImages; i++ ){
            d = _mc.getNextHighestDepth();
            img = new Image(_mc.createEmptyMovieClip("imgId" + d, d), w, h);
            imageContainer.push(img);
      }

      getNextImage();
}
```

The `getNextImage()` method is central to the loading of multiple images for `ImageGrid`. First the `loadCounter` variable is incremented, then the `if` statement looks at this value and checks whether it is less than the total number of images that have been assigned to the `totalImages` variable. If this is the case, then the `_dataProvider` on the image reference is passed a reference to the location to the `_dataProvider[loadCounter]`. The `onImageLoaded()` event handler is invoked in response to a successful image load and the `onLoadComplete` event's being fired. Within the `onImageLoaded()` method, the event listener is removed, the image is then aligned via the

www.it-ebooks.info

alignImage() method, and then finally the getNextImage() method is called again. The most significant aspect here is the onLoadComplete event, which is dispatched when the loadCounter is greater than the totalImages (see Listing 8-14).



LISTING 8-14: Methods used to load multiple images in the ImageGrid

```
private function getNextImage():Void {

      loadCounter++;

      var i:Image = Image(imageContainer[loadCounter]);
      i.addEventListener("onLoadComplete", Delegate.create(this, onImageLoaded));

      if (loadCounter < totalImages) {
            i.dataProvider = _dataProvider[loadCounter];
      } else {
            dispatchEvent({ target:this, type:"onLoadComplete" });
      }
}

private function onImageLoaded():Void {

      var i:Image = Image(imageContainer[loadCounter]);
      i.removeEventListener("onLoadComplete");

      alignImage();
      getNextImage();

}

private function alignImage():Void {

      if (columnCount == columnMax) {
            columnCount = 0;
            rowCount = rowCount + 1;
      }

      if ( rowCount > rowMax) {
            trace("WARNING NOT ENOUGH ROOM IN THE GRID FOR THIS IMAGE!");
      }

      var xPos:Number = (columnCount * imageWidth) + (columnPadding * columnCount );
      var yPos:Number = (rowCount * imageHeight) + (rowPadding*rowCount);

      imageContainer[loadCounter].moveTo(xPos, yPos);
      columnCount = columnCount + 1;

}
```

The drawFrame() method for ImageGrid draws a 2 px–thick frame, with the line color value of 0xEDFF26 around the selected image based on each imageWidth and imageHeight, and then aligns the frame to the highlighted image (see Listing 8-15). The MovieClip container for the frame is added to the view when ImageGrid is constructed; this is a layer above the image displayed.

www.it-ebooks.info



LISTING 8-15: The drawFrame() method for ImageGrid

```
private function drawFrame():Void {
      frame.lineStyle(2, 0xEDFF26, 100, true, "none", "square", "miter", 255);
      frame.moveTo(-1,-1);
      frame.lineTo(imageWidth + 1, -1);
      frame.lineTo(imageWidth + 1, imageHeight + 1);
      frame.lineTo(-1, imageHeight + 1);
      frame.lineTo(-1, -1);
      frame._x = imageContainer[_selectedItem].getX();
      frame._y = imageContainer[_selectedItem].getY();
}
```

To hide the images from view, call the public `hideImages()` method. This loops through each `Image` instance saved in the `imageContainer` and invokes the `setAlpha()` method, setting the alpha value to `0`. The `revealImages()` method does the reverse, setting the alpha value to `100` (see Listing 8-16).



LISTING 8-16: Methods to hide and reveal images in ImageGrid

```
public function hideImages():Void {
      frame._alpha = 0;

      for (var i:Number = 0; i < totalImages; i++ ){
            Image(imageContainer[i]).setAlpha(0);
      }
}

public function revealImages():Void {
      frame._alpha = 100;

      for (var i:Number = 0; i < totalImages; i++ ) {
            Image(imageContainer[i]).setAlpha(100);
      }
}
```

The selected image for the grid can be set via the `selectedItem()` method. The method updates the `_selectedItem` variable and then calls the `drawFrame()` method of the image grid (see Listing 8-17).



LISTING 8-17: The selectedItem() method for ImageGrid

```
public function selectedItem(value:Number):Void {
      _selectedItem = value;
       drawFrame();
}
```

The keys on a device should be mapped to the `nextRow()`, `previousRow()`, `nextColumn()`, and `previousColumn()` methods. Within each method, the `columnCount` and `_selectedItem` variables are used (see Listing 8-18).

www.it-ebooks.info



LISTING 8-18: Methods for navigating the images in ImageGrid

```
public function nextRow():Void {
      var nextImage:Number = _selectedItem + columnCount;
      if (nextImage < totalImages){
            selectedItem = nextImage;
      }
}

public function previousRow():Void {
      var previousImage:Number = _selectedItem – columnCount;
      if (previousImage > 0){
            selectedItem = previousImage;
      }
}

public function nextColumn():Void {
      selectedItem = selectedItem + 1;
      if(selectedItem >= totalImages){
            selectedItem = 0;
      }
}

public function previousColumn():Void {
      selectedItem = selectedItem – 1;
      if(selectedItem < 0){
            selectedItem = 0 + columnMax -1;
      }
}
```

As with the `Image` component, `ImageGrid` has the `moveTo()` method defined so that the whole `ImageGrid` container can be aligned (see Listing 8-19).



LISTING 8-19: The moveTo() method for ImageGrid

```
public function moveTo(x:Number, y:Number ):Void {
      _mc._x = x;
      _mc._y = y;
}
```

The `ImageGrid` and `Image` components will be used as one of the UI components in the `ImageGridMediator`, which will be covered later.

The SharedObject Class

Although this may seem to be an unrelated subject, the `SharedObject` class is introduced here as a core component for the Image Viewer application. It allows for persistent data to a device. This section will demonstrate how data can be retained and when an application closes.

Using shared objects, data can be stored on the device via values set within an application saved by the `.swf` file. This has proved to be a very popular feature of Flash Lite mobile games, where high scores or an application state can be saved and retrieved, providing a better user experience for end users.

www.it-ebooks.info

Local shared objects were introduced as a new feature of Macromedia Flash MX for ActionScript 2.0 and have been a key part of the AS library since then.

For the Image Viewer application, you simply store a reference to the last image selected by the user and then highlight the image when the application restarts.

Shared objects are stored as files as name/value pairs in a `.so` file on the device. One of the downsides to the Flash Lite implementation of `SharedObject` is that it doesn't allow for the sharing of data between different `.swf` files.

The `SharedObject` class has three main methods, each of which will be used in the application:

- `flush()` — saves a shared object to a physical file on the device
- `getLocal()` — returns a reference to the shared object
- `getSize()` — returns the size of the shared object specified

When you create the `SharedObjectProxy` for the Image Viewer, you'll learn how to assign data to the properties of a shared object via the `SharedObject.data` object.

When using `getSize()`, you can actually check that the shared object exists. This method returns the current size of the shared object in bytes. You can then perform a check to see whether your data actually contains a value. This is a useful technique as it ensures that before you assign a value to the data, a variable can properly retrieve data from a shared object.

The `SharedObject` has an `onStatus` event invoked when the event handler is fired from a shared object instance.

When you use the `getMaxSize()` method, the total size of the shared object reserved for the Flash Lite Player for the device is returned.

In addition to the `onStatus` event, you can add the `addListener()` method to the `SharedObject` class to invoke a method using the name of the `SharedObject` you want to specify.

Finally, the `clear()` method removes the shared object from the device.

Now that you are familiar with `MovieClipLoader` and `SharedObject`, let's focus on building the application using PureMVC.

BUILDING THE APPLICATION

In this section you'll explore the code behind the Image Viewer, covering the following aspects of building the application:

- The `.fla` file
- Defining `ApplicationFacade`
- Creating a Model for the Image viewer client
- Creating the Image Viewer's Controller
- Exploring the View's Mediators

Before you begin, be sure that you have access to all the files associated with this chapter.

www.it-ebooks.info

The .fla File

Open the `chapter8_ImageViewer.fla` file in Flash CS4. You should see that on the first frame of the timeline, the call to kickstart the application is defined:




```
import com.wrox.chapter8.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
app.startup(this);
```

code snippet chapter8_ImageViewer.fla

The Library panel contains only three assets: a `SoftKeys` component, a `WaitIndicator` component, and a Separator `MovieClip`. The `SoftKeys` and `WaitIndicator` are covered in Chapter 4. The Separator is simply a decorator graphic for the application, consisting of two lines.

If you publish the file, you should see the Image Viewer load six images via the image grid. Figure 8-2 shows a mockup of the final Image Viewer.



FIGURE 8-2

Defining ApplicationFacade

In this section you'll look at the notifications used in the Image Viewer. You'll also examine the `initializeController()` method and the majority of commands registered in `ApplicationFacade`.



The `startup()` *and* `ApplicationFacade.getInstance()` *methods that are called on the timeline are covered in Chapter 6.*

Defining the Notification Names

The `ApplicationFacade` class has numerous notification names that are used by various parts of the application (see Listing 8-20).



LISTING 8-20: The notification names defined for the Image Viewer ApplicationFacade

```
public static var GET_NEXT_IMAGE:String = "getNextImage";
public static var GET_PREVIOUS_IMAGE:String = "getPreviousImage";
public static var GET_STORED_IMAGE:String = "getStoredImage";
public static var LOAD_IMAGES_XML:String = "loadImagesXML";
public static var LOAD_IMAGES_XML_SUCCESS:String = "loadImagesXMLSuccess";
public static var LOAD_IMAGES_XML_FAILURE:String = "loadImagesXMLFailure";
public static var PREPARE_IMAGES:String = "prepareImages";
public static var PREPARE_IMAGES_COMPLETED:String = "prepareImagesCompleted";
public static var SAVE_IMAGE:String = "saveImage";
```

www.it-ebooks.info

```
public static var SELECT_FIRST_IMAGE:String = "selectFirstImage";
public static var SELECT_IMAGE:String = "selectImage";
public static var SELECTED_INDEX_UPDATED:String = "selectdIndexUpdated";
public static var SET_INDEX:String = "setIndex";
public static var SHARED_OBJECT_LOAD_SUCCESS:String = "sharedObjectLoadSuccess";
public static var SHARED_OBJECT_LOAD_FAILED:String = "sharedObjectLoadFailed";
public static var SHARED_OBJECT_SAVED:String = "sharedObjectSaved";
public static var SHOW_LOADING_PROGRESS:String = "showLoadingProgress";
public static var STARTUP:String = "startup";
public static var UPDATE_SOFT_KEYS:String = "updateSoftKeys";
public static var VIEW_IMAGE_GRID:String = "viewImageGrid";
public static var VIEW_SINGLE_IMAGE:String = "viewSingleImage";
public static var QUIT:String = "quit";
```

Each notification will be covered in more detail later.

The `initializeController()` method registers a portion of the commands with the Controller in the `ApplicationFacade` for the Image Viewer (see Listing 8-21).



LISTING 8-21: Registering commands in the initializeController() method

```
public function initializeController():Void {
      super.initializeController();
      registerCommand(STARTUP, StartupCommand);
      registerCommand(LOAD_IMAGES_XML, LoadImagesXMLCommand);
      registerCommand(PREPARE_IMAGES, PrepareImagesCommand);
      registerCommand(GET_NEXT_IMAGE, GetNextImageCommand);
      registerCommand(GET_PREVIOUS_IMAGE, GetPreviousImageCommand);
      registerCommand(GET_STORED_IMAGE, GetStoredImageCommand);
      registerCommand(VIEW_SINGLE_IMAGE, ViewSingleImageCommand);
      registerCommand(SELECT_FIRST_IMAGE, SelectFirstImageCommand);
      registerCommand(SET_INDEX, SetIndexCommand);
      registerCommand(SELECT_IMAGE, SelectImageCommand);
      registerCommand(QUIT, QuitCommand);
}
```

Again, `StartupCommand` is the first command to be registered in the `initializeController()` method. This is mapped to the `STARTUP` notification name. The `STARTUP` notification is sent in the `startup()` method of `ApplicationFacade`. Both `StartupCommand` and `QuitCommand` have been covered in Chapter 5 and the chapters following it.

In the `initializeController()` method, numerous new notification names are mapped to commands:

- `LOAD_IMAGES_XML` notification is mapped to `LoadImagesXMLCommand`, and when this is sent it loads the image XML data used by the application.
- `PREPARE_IMAGES` is mapped to `PrepareImagesCommand`, and when this command is called it is responsible for ensuring that the images loaded in the XML are saved to `ImageProxy`.
- `VIEW_IMAGE_GRID` is mapped to `ViewImageGridCommand`, and when this notification is sent it is responsible for displaying the images prepared for the application.

www.it-ebooks.info

- `VIEW_SINGLE_IMAGE` is mapped to `ViewSingleImageCommand`, and when this is called it is responsible for displaying a single image in the application.
- `GET_NEXT_IMAGE` is mapped to `GetNextImageCommand`, and when it is called it is responsible for returning the next image from the prepared images.
- `GET_PREVIOUS_IMAGE` is mapped to `GetPreviousImageCommand`, and it returns the previous image from the prepared images.
- `SELECT_FIRST_IMAGE` is mapped to the `SelectFirstImageCommand`, and it is responsible for setting the first image to be displayed.
- `SELECT_IMAGE` is mapped to the `SelectImageCommand`, and it is responsible for handling calls to display a single image.
- `SET_INDEX` is mapped to the `SetIndexCommand`, and it sets the selected image for the image collection.
- `GET_STORED_IMAGE` is mapped to `GetStoredImageCommand`, and this command is responsible for retrieving the reference image stored on the device.

You'll cover each of these commands later in this chapter. Next up is creating the model.

Creating a Model for the Image Viewer

In this section you'll examine the model defined for the application (see Figure 8-3).

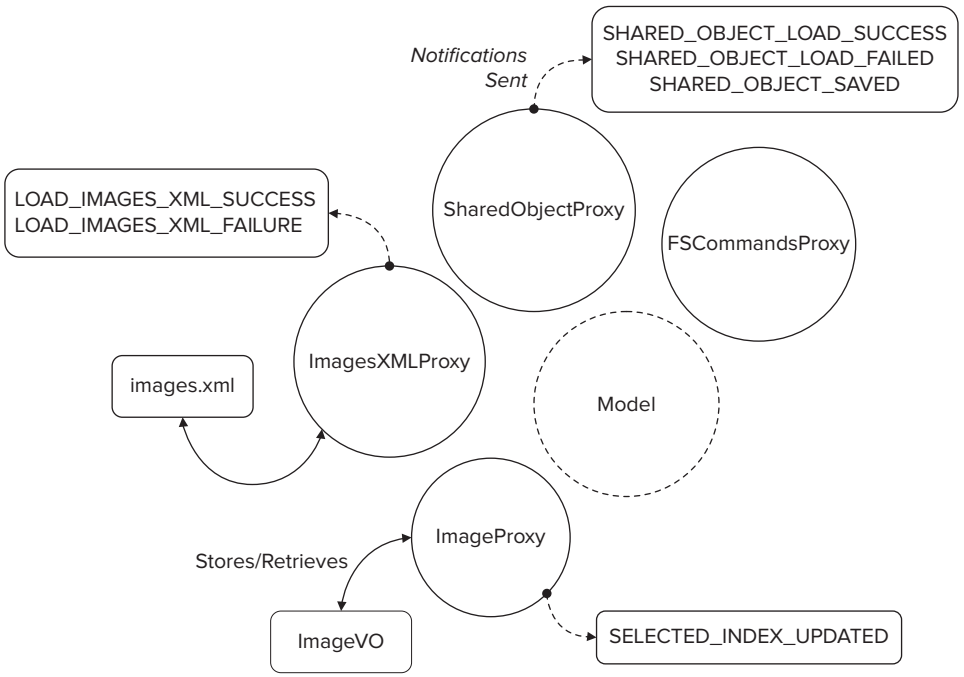


FIGURE 8-3

www.it-ebooks.info

In Figure 8-3, you can see three new proxy classes that are used for the Image Viewer application. Each of these has its own responsibility:

- `ImagesXMLProxy` provides methods for loading and parsing the Images XML.
- `ImageProxy` provides methods for navigating the image data retrieved from the Images XML.
- `SharedObjectProxy` provides methods for storing and retrieving the shared object data to the device.

`FSCommandsProxy` is essentially the same proxy as covered in earlier chapters and is not covered here.

The Project panel shown in Figure 8-4 indicates the file system layout for the Image Viewer. The figure shows each of the files under the model directory.

Before looking at the new proxy classes in detail, first take a look at the `ImageVO`.

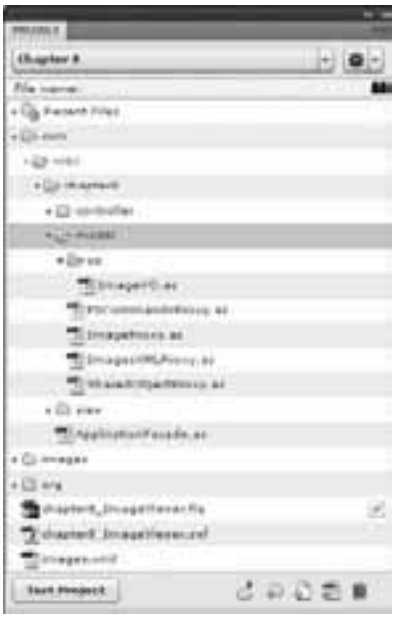


FIGURE 8-4

ImageVO.as

The model stores the image data as image value objects called `ImageVO` (see Listing 8-22).



LISTING 8-22: The ImageVO class

```
class com.wrox.chapter8.model.vo.ImageVO {

      public var index:Number;
      public var id:String;
      public var title:String;
      public var thumbPath:String;
      public var largePath:String;

      public function ImageVO(){

      }
}
```

Each of the public variables defined in `ImageVO` is taken from our custom `images.xml` file (see Listing 8-23).



LISTING 8-23: The images.xml file

```
<images title="Brawn GP" subTitle="">
      <img id="1" title="BGP 001, Nose"
           thumbPath="images/01_sml.png" largePath="images/01_lrg.gif" />
```

continues

www.it-ebooks.info

LISTING 8-23 *(continued)*

```
        <img id="2" title="Rubens Barrichello, Qualifying"
             thumbPath="images/02_sml.png" largePath="images/02_lrg.gif" />
        <img id="3" title="Circuit de Catalunya, Barcelona, Race Start"
             thumbPath="images/03_sml.png" largePath="images/03_lrg.gif" />
        <img id="4" title="Rubens Barrichello & Jenson Button, Racing in China"
             thumbPath="images/04_sml.png" largePath="images/04_lrg.gif" />
        <img id="5" title="Rubens Barrichello, Racing in China"
             thumbPath="images/05_sml.png" largePath="images/05_lrg.gif" />
        <img id="6" title="The BGP 001, Super Quick!"
             thumbPath="images/06_sml.png" largePath="images/06_lrg.gif" />
</images>
```

In the XML you can see where each of the variables defined in `ImageVO` is taken from. `ImageVO` has several properties: an `id` string variable to represent the image uniquely, a `title` variable, a `thumbPath` for the smaller image thumbnail reference, and a `largePath` for the large image reference. `ImageVO` also has an `index` variable number to represent the image and its position in the collection returned in the XML.

Loading the Image XML Data

In this section you'll look at creating the `ImagesXMLProxy` class used for loading image data from the XML.

ImagesXMLProxy.as

The `ImagesXMLProxy` will load the XML used in the application.

1. Create the skeleton for the proxy, importing `ApplicationFacade`, `Delegate`, `XPathAPI`, `ImageVO`, and `Proxy` classes and the `IProxy` interface (see Listing 8-24).



LISTING 8-24: Creating the ImagesXMLProxy

```
import com.wrox.chapter8.ApplicationFacade;
import com.wrox.chapter8.model.vo.ImageVO;

import mx.utils.Delegate;
import mx.xpath.XPathAPI;

import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter8.model.ImagesXMLProxy extends Proxy implements IProxy {

      public static var NAME:String = "ImagesXMLProxy";

      public function XMLProxy(data:Object){
            super(NAME, data);
      }
}
```

www.it-ebooks.info

2. Add two private variables: `xml` to capture the `XML` data, and `url` to reference the path to the `images.xml` file. Also, create stubs for a public `load()` method and two private methods, `onImageXMLLoadComplete()` and `parseXML()` (see Listing 8-25).



LISTING 8-25: Adding two private variables to ImagesXMLProxy

```
class com.wrox.chapter8.model.ImagesXMLProxy extends Proxy implements IProxy {

      public static var NAME:String = "ImagesXMLProxy";

      private var xml:XML;
      private var url:String = "images.xml";

      public function XMLProxy(data:Object) {
            super(NAME, data);
      }

      public function load():Void {}

      private function onImageXMLLoadComplete():Void {}
      private function parseXML():Void {}

}
```

3. Complete the `load()` method, making a call to the `xml.load()` method and passing in a reference to the `images.xml` file (see Listing 8-26).



LISTING 8-26: The load() method for ImagesXMLProxy

```
public function load():Void {
      xml = new XML();
      xml.onLoad = Delegate.create(this, onImageXMLLoadComplete);
      xml.ignoreWhite = true;
      xml.load(url);
}
```

4. Complete the `onImageXMLLoadComplete()` method. Here you need to define the `success` parameter, which you can then use to detect whether the `XML` successfully loaded. If the load was successful, you can then call the `parseXML()` method of the proxy and pass in a reference to `xml.firstChild`, which is an `XMLNode`.

   You also need to send the `LOAD_IMAGES_XML_SUCCESS` notification. If the proxy fails to load the XML, then `LOAD_IMAGES_XML_FAILURE` notification is sent (see Listing 8-27).

www.it-ebooks.info



LISTING 8-27: The onImageXMLLoadComplete() method for ImagesXMLProxy

```
public function load():Void {
      xml = new XML();
      xml.onLoad = Delegate.create(this, onImageXMLLoadComplete);
      xml.ignoreWhite = true;
      xml.load(url);
}

private function onImageXMLLoadComplete(success:Boolean):Void {
      if (success) {
            parseXML(xml.firstChild);
            sendNotification(ApplicationFacade.LOAD_IMAGES_XML_SUCCESS);
      } else {
            sendNotification(ApplicationFacade.LOAD_IMAGES_XML_FAILURE);
      }
}
```

5. Complete the `parseXML()` method. As mentioned in the previous step, `parseXML()` requires that the first `XMLNode` be a parameter. This is returned from the initial `xml.load()` call. The `parseXML()` method also uses `parseItemXMLNode()`; create the stub for this new method also (see Listing 8-28).



LISTING 8-28: The parseXML() method for ImagesXMLProxy

```
private function parseXML(xmlNode:XMLNode):Void {

      var dataArray:Array = [];
      var xmlNodeList:Array = XPathAPI.selectNodeList(xmlNode, "images/img");

      for(var i:Number=0; i < xmlNodeList.length; i++){
            dataArray.push(parseItemXMLNode(xmlNodeList[i], i));
      }

      setData(dataArray);
}

private function parseItemXMLNode(node:XMLNode, index:Number):Object {

}
```

6. Complete the `parseItemXMLNode()` method. Here's where you need to assign each of the attributes from each `img childNode` (see Listing 8-29).



LISTING 8-29: The parseItemXMLNode() method for ImagesXMLProxy

```
private function parseItemXMLNode(node:XMLNode, index:Number):Object {
      var vo:ImageVO = new ImageVO();
      var childNode:XMLNode;
```

www.it-ebooks.info

```
            childNode = XPathAPI.selectSingleNode(node, "img");
            vo.index = index;
            vo.id = childNode.attributes.id;
            vo.title = childNode.attributes.title;
            vo.thumbPath = childNode.attributes.thumbPath;
            vo.largePath = childNode.attributes.largePath;

            return vo;
      }
```

This completes the `ImagesXMLProxy`. The call to load the XML via the public `load()` method will be covered in the `LoadImagesXMLCommand` section later in this chapter.

Handling Image Collections

In this section you'll look at creating `ImageProxy.as`, used for handling image collections.

ImageProxy.as

The `ImageProxy` class essentially keeps track of which image is selected. It also needs to return image data to the rest of the application. The class will need to provide several getter methods for retrieving the instances or properties of `ImageVO`, including:

- `smallImagePaths()` — to return an array of strings representing each of the `thumbPath` properties on each `ImageVO` object retrieved from the `XML`; this is used to retrieve the small image paths for the image grid
- `selectedLargeImagePath()` — to return the `largePath` of the selected `ImageVO` object
- `selectedTitle()` — to return the `title` property of the selected image path of the current selected image
- `nextImage()` — to return the next available `ImageVO`
- `previousImage()` — to retrieve the previously available `ImageVO`

Follow these steps to create the `ImageProxy.as`:

1. Create the skeleton for `ImageProxy.as` (see Listing 8-30).



LISTING 8-30: Creating the ImageProxy class

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

import com.wrox.chapter8.model.vo.ImageVO;

class com.wrox.chapter8.model.ImageProxy extends Proxy implements IProxy {

      public static var NAME:String = "ImageProxy";

      public function ImageProxy(data:Object){
            super(NAME, data);
      }
}
```

www.it-ebooks.info

2. Define the `init()` method to initialize the `ImageProxy` class. To keep track of which image is selected in the Image Viewer, the `_selectedIndex` variable is used, and when the proxy is initialized the variable is set to `0`. The `_dataProvider` object is also used. To use the `_dataProvider` later, add the data set to this variable in the `ImageProxy` constructor (see Listing 8-31).



LISTING 8-31: The init() method for ImageProxy

```
public function ImageProxy(data:Object) {
      super(NAME, data);
}

public function init():Void {
      _dataProvider = getData();
      _selectedIndex = 0;
}
```

3. Underneath the `init()` method, create the stubs for the public methods `selectedImage()`, `nextImage()`, `previousImage()`, and then for `selectedIndex()`. You will complete them shortly (see Listing 8-32).



LISTING 8-32: The stubs for nextImage(), previousImage(), selectedImage(), and selectedIndex() methods in ImageProxy

```
public function init():Void {
      _dataProvider = getData();
      _selectedIndex = 0;
}

public function nextImage():Void {}

public function previousImage():Void {}

public function get selectedImage():ImageVO {}

public function get selectedIndex():Number {}

public function set selectedIndex():Void {}
```

4. Complete the getter and setter methods for `selectedIndex()` (see Listing 8-33).



LISTING 8-33: The selectedIndex() method for ImageProxy

```
public function get selectedIndex():Number {
      return _selectedIndex;
}

public function set selectedIndex(index:Number):Void {
      _selectedIndex = index;
}
```

www.it-ebooks.info

5. Complete the `selectedImage()` method. The method should return an `ImageVO` from the `_dataProvider` array using the `_selectedIndex` variable (see Listing 8-34).



LISTING 8-34: The selectedImage() method for ImageProxy

```
public function get selectedImage():ImageVO {
      return _dataProvider[_selectedIndex];
}
```

6. Complete `nextImage()` and `previousImage()` methods. Here you should see that both methods send the `SELECTED_INDEX_UPDATED` notification along with an `ImageVO` returned from using `selectedImage()` (see Listing 8-35).



LISTING 8-35: The nextImage() and previousImage() methods for ImageProxy

```
public function nextImage():Void {
      _selectedIndex++;

      if (_selectedIndex >= _dataProvider.length){
            _selectedIndex = 0;
      }

      sendNotification(ApplicationFacade.SELECTED_INDEX_UPDATED, selectedImage);
}

public function previousImage():Void {
      _selectedIndex--;

      if (_selectedIndex <= -1){
            _selectedIndex = _dataProvider.length-1;
      }

      sendNotification(ApplicationFacade.SELECTED_INDEX_UPDATED, selectedImage);
}
```

7. Complete the `smallImagePaths()` and `selectedLargeImagePath()` getter methods underneath `init()` (see Listing 8-36).



LISTING 8-36: The selectedLargeImagePath() and smallImagePaths() method for ImageProxy

```
public function init():Void {
      _dataProvider = getData();
      _selectedIndex = 0;
}

public function get selectedLargeImagePath():String {
      var image:ImageVO = selectedImage;
      return image.largePath;
}
```

continues

www.it-ebooks.info

LISTING 8-36 *(continued)*

```
public function get smallImagePaths():Array {
      var i:Number;
      var thumbPaths:Array = new Array();
      var imageVO:ImageVO;

      for (i = 0; i < _dataProvider.length; i++) {
            imageVO = ImageVO(_dataProvider[i]);
            thumbPaths.push(imageVO.thumbPath);
      }

      return thumbPaths;
}
```

8. Create the method for retrieving the image titles (see Listing 8-37).



LISTING 8-37: The selectedTitle method for ImageProxy

```
public function get selectedTitle():String {
      var image:ImageVO = selectedImage;
      return image.title;
}
```

This completes the `ImageProxy`. The class will be called by `PrepareImagesCommand` before it renders the image grid. The next proxy we'll cover is `SharedObjectProxy`.

Storing Image Data on the Device

In this section you'll examine the `SharedObject` implementation in the `SharedObjectProxy` class.

SharedObjectProxy.as

The `SharedObjectProxy` class, also shown in Figure 8-3, needs to retrieve the last image selected by the user so that the image grid knows which image to highlight when the application launches.

Three methods are used in the `SharedObjectProxy` class:

- `init()` — to initialize a `SharedObject` instance
- `onLoadComplete()` — to handle data retrieved from the local `SharedObject` instance
- `saveSO()` — to save new image data references to the `SharedObject` instance

The proxy sends three notifications on launch to the rest of the application: `SHARED_OBJECT_LOAD_SUCCESS` when the shared object has loaded; then `SHARED_OBJECT_LOAD_FAILED`, when the shared object fails to load; and then `SHARED_OBJECT_SAVED`, when the shared object is saved.

www.it-ebooks.info

1. Create the skeleton for `SharedObjectProxy`, including the stubs for `init()`, `onLoadComplete()`, and `saveSO()`. Import the `Delegate` class for this proxy and ensure that you add a private variable instance for the `SharedObject` reference above the `SharedObjectProxy` constructor (see Listing 8-38).



LISTING 8-38: Creating the SharedObjectProxy class

```
import com.wrox.chapter8.ApplicationFacade;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter8.model.SharedObjectProxy extends Proxy implements IProxy {

      public static var NAME:String = "SharedObjectProxy";

      private var sObj:SharedObject;

      public function SharedObjectProxy(data:Object) {
            super(NAME, data);
      }

      public function init():Void {}

      private function onLoadComplete():Void {}

      public function saveSO():Void {}

}
```

2. Complete the `init()` method, specifying the parameter `referenceStr`. Here you also create a handler for the static `SharedObject.addlistener()` method, calling `onLoadComplete()`; next, use the `SharedObject.getLocal()` method to retrieve the `SharedObject` from local storage (see Listing 8-39).



LISTING 8-39: The init() method for SharedObjectProxy

```
public function init(referenceStr:String):Void {
      SharedObject.addListener(referenceStr, Delegate.create(this, onLoadComplete));
      sObj = SharedObject.getLocal(referenceStr);
}
```

www.it-ebooks.info

3. Complete the `onLoadComplete()` method. Here you need to define `dataObj`, a `SharedObject` parameter, returned when the retrieval of the object through `getLocal()` is complete. This method needs to send one of two notifications. If there is stored data retrieved, `SHARED_OBJECT_LOAD_SUCCESS` is sent, whereas if there is no stored data, `SHARED_OBJECT_LOAD_FAILED` is sent. When there is data available, you can send the `dataObj.data` object with the notification. This object is also saved to the proxy via the `setData()` method (see Listing 8-40).



LISTING 8-40: The onLoadComplete() method for SharedObjectProxy

```
private function onLoadComplete(dataObj:SharedObject):Void {
      if (dataObj.getSize() != 0) {
            setData(dataObj.data);
            sendNotification(ApplicationFacade.SHARED_OBJECT_LOAD_SUCCESS,
                                                            dataObj.data, "Object" );
      } else {
            sendNotification(ApplicationFacade.SHARED_OBJECT_LOAD_FAILED);
      }
}
```

4. Complete the public `saveSO()` method. This method needs to take in an object as a parameter, called `dataObj`. The object's properties should then be saved to our class property `sObj`, and then the `flush()` method is called to save the new values (see Listing 8-41).



LISTING 8-41: The saveSO() method for SharedObjectProxy

```
public function saveSO(dataObj:Object):Void {
      for(var i:String in dataObj){
            sObj.data[i] = dataObj[i];
      }

      sObj.flush();
      sendNotification(ApplicationFacade.SHARED_OBJECT_SAVED);
}
```

You've now covered each of the methods for `SharedObjectProxy`. Of course, now with each of the Proxy classes completed, later the `ModelPrepCommand` will be outlined.

Exploring the View

In this section you'll explore the view elements used for the Image Viewer application (see Figure 8-5).

www.it-ebooks.info

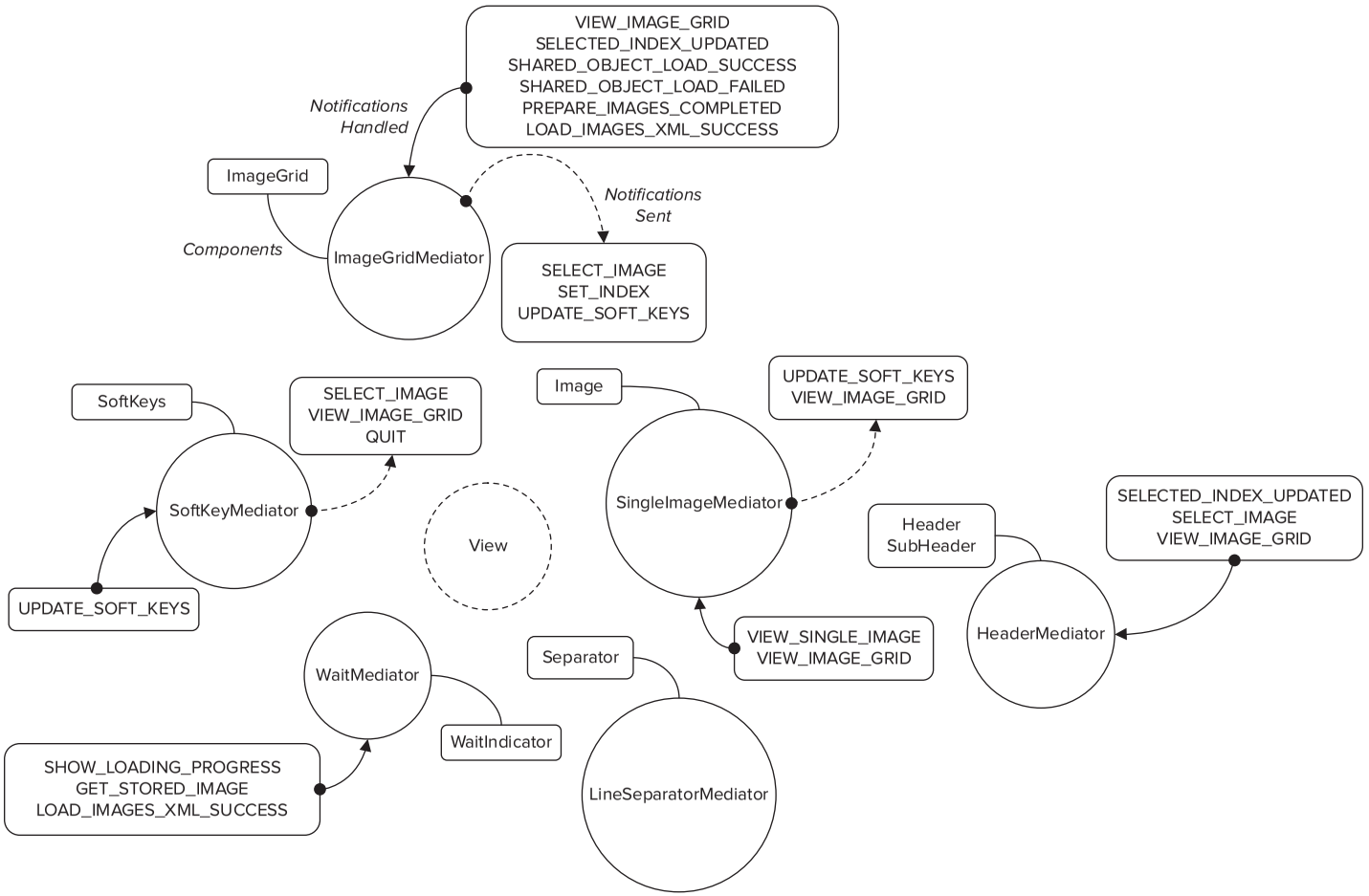


FIGURE 8-5

www.it-ebooks.info

In Figure 8-5, you can see the mediators used in the Image Viewer application:

- `SingleImageMediator` displays the main image, once a thumbnail has been selected from the `ImageGridMediator`
- `ImageGridMediator` displays six thumbnail images taken from the `image.xml` file
- `HeaderMediator` displays and updates the title for the highlighted image in the `ImageGridMediator` view and the selected image in the `SingleImageMediator`
- `SoftKeyMediator` displays the soft key labels and sends notes assigned to the keys when they are pressed
- `WaitMediator` displays the wait indicator to show that images are loading when the application starts
- `LineSeparatorMediator` displays a line in between `HeaderMediator` and `ImageGridMediator`

The Project panel shown in Figure 8-6 indicates the file system layout for the image viewer, showing each of the files under the view directory.



FIGURE 8-6

Viewing a Single Image

Open the `SingleImageMediator` source file to view the code in full.

SingleImageMediator.as

`SingleImageMediator` is essentially responsible for showing a single image in the application when the user selects an item from the Image Grid by pressing the Enter key. The mediator responds to two notifications, `VIEW_SINGLE_IMAGE` and `VIEW_IMAGE_GRID`.

`SingleImageMediator` uses a single `Image` component to render selected images. The `onRegister()` method is where you'll find that the class instantiates a new `Image` instance called `img`. Images are loaded into the view via the `loadImage()` method. This is actually achieved through retrieval of the mediator in the `ViewSingleImageCommand` and calling the `loadImage()` method through here (see Listing 8-42).



LISTING 8-42: The loadImage() method for SingleImageMediator

```
public function loadImage(imagePath:String):Void {
      img.dataProvider = imagePath;
      updateSoftKeys();
}
```

The `updateSoftKeys()` method for the mediator simply sets the left soft key `label` to the string "Grid" for the `SoftKeyMediator` (see Listing 8-43). When this key is pressed, `SoftKeyMediator` sends the `VIEW_IMAGE_GRID` notification, at which point the `destroy()` method is called to remove the large image from view.

www.it-ebooks.info



LISTING 8-43: The updateSoftKeys() method for SingleImageMediator

```
private function updateSoftKeys():Void  {
      var msk:Object = {label:"", note:""};
      var lsk:Object = {label:" Grid ", note:ApplicationFacade.VIEW_IMAGE_GRID};
      var rsk:Object = {label:" ", note:""};

      sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
}
```

Viewing Multiple Images

Open the `ImageGridMediator` source file to view the code in full.

ImageGridMediator.as

`ImageGridMediator` displays each of the thumbnails found in the `image.xml` file and registers six notifications with the view (see Listing 8-44).



LISTING 8-44: The listNotificationInterests method for ImageGridMediator

```
public function listNotificationInterests():Array {
      return [ ApplicationFacade.VIEW_IMAGE_GRID,
               ApplicationFacade.SELECTED_INDEX_UPDATED,
               ApplicationFacade.SHARED_OBJECT_LOAD_SUCCESS,
               ApplicationFacade.SHARED_OBJECT_LOAD_FAILURE,
               ApplicationFacade.PREPARE_IMAGES_COMPLETED,
               ApplicationFacade.LOAD_IMAGES_XML_SUCCESS ];
}
```

`ImageGridMediator` includes the following functionality:

- Key events handler to handle navigation of the image grid
- `ImageGrid` component to render the thumbnails for each image in the XML

The `handleNotification()` method handles each of the notifications listed in `listNotificationInterests()`. The `VIEW_IMAGE_GRID` notification is mapped to `ViewImageGridCommand` and sets the image data through `dataProvider` on the `ImageGrid` component when the notification is sent. The `SELECTED_INDEX_UPDATED` notification, which isn't actually mapped to a command, is used to update the selected item on the `ImageGrid` shown here by selecting `imgs.selectedItem` on the component (see Listing 8-45).



LISTING 8-45: The handleNotification() method for ImageGridMediator

```
public function handleNotification(note:INotification):Void {
      switch (note.getName()) {
            case ApplicationFacade.LOAD_IMAGES_XML_SUCCESS:
                  sendNotification(ApplicationFacade.PREPARE_IMAGES);
            break;
            case ApplicationFacade.PREPARE_IMAGES_COMPLETED:
```

continues

www.it-ebooks.info

LISTING 8-45 *(continued)*

```
                updateSoftKeys();
                imgs.dataProvider = note.getBody()[0];
                activateKeyListener();
            break;
            case ApplicationFacade.SELECTED_INDEX_UPDATED:
                imageVO = ImageVO(note.getBody());
                imgs.selectedItem(imageVO.index);
            break;
            case ApplicationFacade.VIEW_IMAGE_GRID:
                updateSoftKeys();
                imgs.reveal();
                activateKeyListener();
            break;
            case ApplicationFacade.SHARED_OBJECT_LOAD_SUCCESS:
                var index:Number = note.getBody().selectedIndex
                imgs.selectedItem(index);
                sendNotification(ApplicationFacade.SELECT_FIRST_IMAGE, index);
            break;
            case ApplicationFacade.SHARED_OBJECT_LOAD_FAILED:
                sendNotification(ApplicationFacade.SELECT_FIRST_IMAGE);
            break;
        }
    }
```

Finally, handle the `VIEW_IMAGE_GRID` notification by calling `imgs.reveal()` so you know when to display the images in the grid.

In the `onRegister()` method for the `ImageGridMediator`, you instantiate a new `ImageGrid` component called `imgs`.

Here the image grid has each thumb image dimension as 90 × 60 px (width set to 90 px and height set to 60 px). Our image grid has 3 columns and 2 rows and also has horizontal and vertical padding of 11 px between each of the images. Notice also that the image grid has the `addEventListener()` method set for `onLoadComplete` event which, if you remember from earlier, is when the image grid has fully loaded. The `onLoadComplete()` method is where the `GET_STORED_IMAGE` notification can be sent. This should let you know which image was the last selected when the app restarts (see Listing 8-46).



LISTING 8-46: The onRegister() and onLoadComplete() methods for ImageGridMediator

```
public function onRegister():Void {
        imgs = new ImageGrid( _mc, 90, 60, 3, 3, 11 );
        imgs.moveTo(15, 68);
        imgs.addEventListener("onLoadComplete", Delegate.create(this, onLoadComplete));
}

public function onLoadComplete():Void {
        sendNotification(ApplicationFacade.GET_STORED_IMAGE);
}
```

In `onRegister()` you can also create an event handler for the key listener's `onKeyDown` event called `onKeyDownHandler()`. The complete `onRegister()` method is shown in Listing 8-47.

www.it-ebooks.info


Available for download on Wrox.com

LISTING 8-47: The keyListener object for ImageGridMediator

```
public function onRegister():Void {
      imgs = new ImageGrid( _mc, 90, 60, 3, 3, 11 );
      imgs.moveTo(15, 68);
      imgs.addEventListener("onLoadComplete", Delegate.create(this,
onLoadComplete));

      keyListener = new Object();
      keyListener.onKeyDown = Delegate.create(this, onKeyDownHandler);
}
```

Returning to the `onKeyDownHandler()`, remember that this is the key handler that is also activated and then deactivated when the mediator is no longer in use. The `onKeyDownHandler()` method needs to capture five key presses from the user. Presses of the Up and Down keys of the device need to be captured so that the user can navigate through the rows of the image grid. Presses of the Left and Right keys need to be captured for the user to navigate through columns. The directional keys send the `SET_INDEX` notification, along with the `selectedItem` from the `ImageGrid` instance. The Enter key or Select is used to view the selected image; this key press, along with the Page Down key, sends the `SELECT_IMAGE` notification along with the `ImageVO` instance (see Listing 8-48).


Available for download on Wrox.com

LISTING 8-48: The onKeyDownHandler() method for ImageGridMediator

```
public function onKeyDownHandler():Void {
      switch (Key.getCode()) {
            case Key.UP:
                  imgs.previousRow();
                  sendNotification(ApplicationFacade.SET_INDEX, imgs.selectedItem);
            break;
            case Key.DOWN:
                  imgs.nextRow();
                  sendNotification(ApplicationFacade.SET_INDEX, imgs.selectedItem);
            break;
            case Key.LEFT:
                  imgs.previousColumn();
                  sendNotification(ApplicationFacade.SET_INDEX, imgs.selectedItem);
            break;
            case Key.RIGHT:
                  imgs.nextColumn();
                  sendNotification(ApplicationFacade.SET_INDEX, imgs.selectedItem);
            break;
            case Key.ENTER:
                  imgs.hideImages();
                  deactivateKeyListener();
                  sendNotification(ApplicationFacade.SELECT_IMAGE, imageVO);
            break;
            case Key.PGDN:
                  imgs.hideImages();
                  deactivateKeyListener();
                  sendNotification(ApplicationFacade.SELECT_IMAGE, imageVO);
            break;
      }
}
```

www.it-ebooks.info

You've now covered the key aspects of `ImageGridMediator`. Both the `SET_INDEX` and `SELECT_IMAGE` notifications, and associated commands, are discussed in more detail later.

Waiting for Images to Load

The Image Viewer uses the `WaitMediator` to display the `WaitIndicator` component covered in Chapter 4. Because images are local to the application, chances are you won't necessarily see the indicator showing.

WaitMediator.as

`WaitMediator` has a similar structure to the other mediators in the application. It is responsible for displaying the `WaitIndicator` component symbol as an indication that some process is currently taking place in the application. In the `handleNotification()` method of the mediator, three notifications are handled.

First the `SHOW_LOADING_PROGRESS` notification displays the indicator by calling the private `showIndicator()` method of the mediator. A conditional statement checks to see if `waitIndicator` has already been created. The stage of the movie is used to calculate and align the mediator by using the `Stage.width` and `Stage.height` properties (see Listing 8-49).



LISTING 8-49: The showIndicator() method for WaitMediator

```
public function showIndicator():Void {
      if (!waitIndicator) {
            waitIndicator = _component.attachMovie("WaitIndicator", "waitIndicator",
_component.getNextHighestDepth() );
            waitIndicator._shape = "circle";
            waitIndicator._x = 0 + (Stage.width / 2) – (30/2);
            waitIndicator._y = 0 + (Stage.height / 2) – (30/2);
      }
}
```

The `IMAGE_XML_LOADING_SUCCESS` notification is also handled to let the mediator know when to remove the indicator from view. It does this by calling the `removePreloader()` method (see Listing 8-50).



LISTING 8-50: The removeIndicatorPreloader method for WaitMediator

```
private function removeIndicator():Void {
      waitIndicator.removeMovieClip();
}
```

`handleNotification()` has each of the notifications listened to by the mediator defined (see Listing 8-51).

www.it-ebooks.info



LISTING 8-51: The handleNotification() method for WaitMediator

```
public function handleNotification(note:INotification):Void {
      switch (note.getName()) {
            case ApplicationFacade.SHOW_LOADING_PROGRESS:
                  showIndicator(_component);
            break;
            case ApplicationFacade.GET_STORED_IMAGE:
                  showIndicator(_component);
            break;
            case ApplicationFacade.SHARED_OBJECT_LOAD_SUCCESS:
                  removeIndicator();
            break;
            case ApplicationFacade.LOAD_IMAGES_XML_SUCCESS:
                  removeIndicator();
            break;
      }
}
```

HeaderMediator.as

`HeaderMediator` contains text elements for the selected image title for the image grid and single image grid. It uses custom `Header` and `SubHeader` components that essentially display the text. Looking at the `handleNotification()` method for the `HeaderMediator`, you can determine which notifications the mediator is interested in. The text for each component is set from the title property of `ImageVO`, which is retrieved in the notification body (see Listing 8-52).



LISTING 8-52: The handleNotification() method for HeaderMediator

```
public function handleNotification(note:INotification):Void {
      switch (note.getName()){
            case ApplicationFacade.SELECTED_INDEX_UPDATED:
                  imageVO = ImageVO(note.getBody());
                  subHeader.setText((imageVO.index + 1) + ". " + imageVO.title);
            break;
            case ApplicationFacade.SELECT_IMAGE:
                  imageVO = ImageVO(note.getBody());
                  header.setText((imageVO.index + 1) + ". " + imageVO.title);
                  subHeader.setText("");
            break;
            case ApplicationFacade.VIEW_IMAGE_GRID:
                  header.setText("IMAGE VIEWER");
            break;
      }
}
```

You now have covered each of the mediators used in the Image Viewer.

Creating the Image Viewer's Controller

In this section you'll go through the remaining commands that are used for the Controller in the Image Viewer (see Figure 8-7).

www.it-ebooks.info

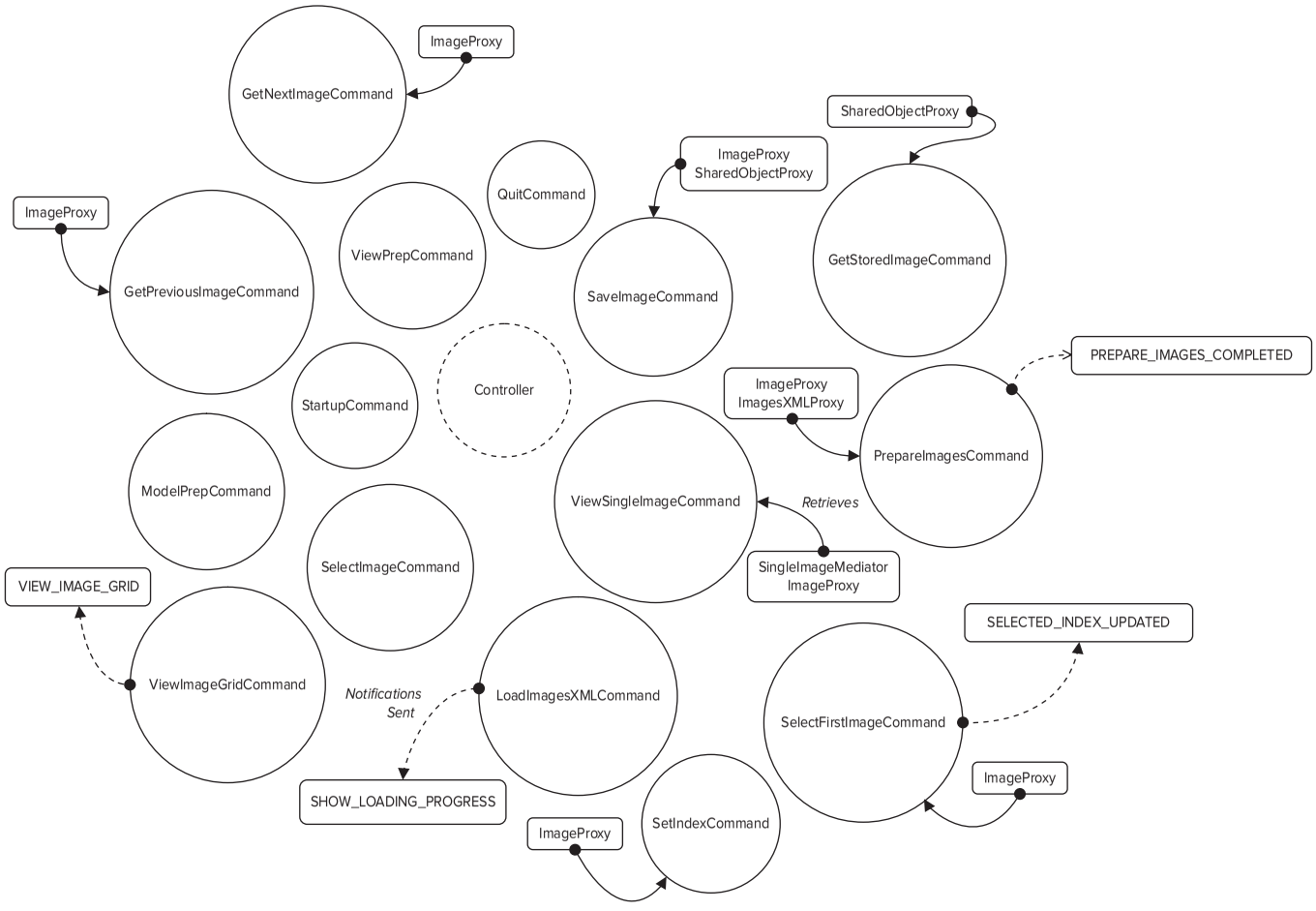


FIGURE 8-7

www.it-ebooks.info

Each command has a simple `execute()` method that carries out a request for the application.

In this section you'll cover:

- Preparing the application
- Loading the images
- Navigating through the images
- Saving a reference to the selected image
- Displaying the image grid
- Displaying a single image

The Project panel shown in Figure 8-8 indicates the file system layout for the image viewer, showing each of the files under the controller directory.



FIGURE 8-8

Preparing the Application

First, you need to prepare the application via `ModelPrepCommand` and `ViewPrepCommand`.

ModelPrepCommand.as

`ModelPreCommand` simply registers the four proxy classes used in the application (see Listing 8-53).

Available for download on Wrox.com

LISTING 8-53: The execute method for ModelPrepCommand

```
public function execute(note:INotification):Void {
      facade.registerProxy(new FSCommandsProxy());
      facade.registerProxy(new SharedObjectProxy());
      facade.registerProxy(new ImagesXMLProxy());
      facade.registerProxy(new ImageProxy());
}
```

ViewPrepCommand.as

`ViewPrepCommand` registers six mediators and also sends the `LOAD_XML` notification (see Listing 8-54).

Available for download on Wrox.com

LISTING 8-54: The execute method for ViewPrepCommand

```
public function execute(note:INotification):Void {
      var mc:MovieClip = MovieClip(note.getBody());

      facade.registerMediator(new ImageGridMediator(mc));
      facade.registerMediator(new SingleImageMediator(mc));
```

continues

www.it-ebooks.info

LISTING 8-54 *(continued)*

```
        facade.registerMediator(new LineSeparatorMediator(mc));
        facade.registerMediator(new SoftKeyMediator(mc));
        facade.registerMediator(new HeaderMediator(mc));
        facade.registerMediator(new WaitMediator(mc));

        sendNotification(ApplicationFacade.LOAD_XML);
    }
```

Loading the Images

Next, you take a look at how to load the images via the `LoadImagesXMLCommand`.

LoadImagesXMLCommand.as

The `execute()` method of `LoadImagesXMLCommand` is where the call to load the `images.xml` file is initialized. Earlier you followed the steps to create `ImagesXMLProxy`, which exposes the public `load()` method. Here you'll see that the `SHOW_LOADING_PROGRESS` notification is sent to display an indeterminable progress. This is one of the notifications handled in `WaitMediator` (see Listing 8-55).



LISTING 8-55: The execute method for LoadImagesXMLCommand

```
public function execute(note:INotification):Void {
        var imagesXML:ImagesXMLProxy;
        imagesXML = ImagesXMLProxy(facade.retrieveProxy(ImagesXMLProxy.NAME));
        imagesXML.load();
        sendNotification(ApplicationFacade.SHOW_LOADING_PROGRESS);
    }
```

PrepareImagesCommand.as

`PrepareImagesCommand` is called by `ImageGridMediator` after the successful load of the `images.xml` file. First the command needs to retrieve data loaded in `ImagesXMLProxy`, which then needs to be stored and managed by `ImageProxy`. The `getData()` method is called on the `ImagesXMLProxy` instance and then assigned to `imgProxy` via the `setData()` call (see Listing 8-56).



LISTING 8-56: The execute method for PrepareImagesCommand

```
public function execute(note:INotification):Void {
        var imagesXML:ImagesXMLProxy;
        imagesXML = ImagesXMLProxy(facade.retrieveProxy(ImagesXMLProxy.NAME));

        var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));
        imgProxy.setData(imagesXML.getData());
        imgProxy.init();
    }
```

Once `ImageProxy` is initialized, the `PREPARE_IMAGES_COMPLETED` notification can be sent along with the small image paths (see Listing 8-57).

www.it-ebooks.info



LISTING 8-57: The execute method for PrepareImagesCommand

```
public function execute(note:INotification):Void {
      var imagesXML:ImagesXMLProxy;
      imagesXML = ImagesXMLProxy(facade.retrieveProxy(ImagesXMLProxy.NAME));

      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));
      imgProxy.setData(imagesXML.getData());
      imgProxy.init();

      sendNotification(ApplicationFacade.PREPARE_IMAGES_COMPLETED,
                                                  imgProxy.smallImagePaths);

}
```

Navigating through the Images

Next, you need to be able to navigate through the images.

GetNextImageCommand.as and GetPreviousImageCommand.as

To navigate through the image grid, use these two commands. `GetNextImageCommand`, which retrieves `ImageProxy`, calls the `nextImage()` method to increment the currently selected image (see Listing 8-58).



LISTING 8-58: The execute method for GetNextImageCommand

```
public function execute(note:INotification):Void {
      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));
      imgProxy.nextImage();
}
```

The `GetPreviousImageCommand` calls the `previousImage()` method on the `ImageProxy` instance `imgProxy` (see Listing 8-59).



LISTING 8-59: The execute method for GetPreviousImageCommand

```
public function execute(note:INotification):Void {
      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));
      imgProxy.previousImage();
}
```

Originally, these two commands were assigned to key presses. For the "left" key press `GetPreviousImageCommand` was used, and for the "right" key press `GetNextImageCommand` was used. Later it was decided to use `SetIndexCommand` as it was more flexible. These two commands, however, would be better suited for the single image view. So one challenge would be to add this functionality to the Image Viwer.

www.it-ebooks.info

Saving and Retrieving a Reference to a Selected Image

Finally, you need to save and then be able to retrieve selected images.

SaveImageCommand.as

The `execute()` method of `SaveImageCommand` is where the details of the currently selected image are stored to the device via `SharedObject`. First the object called `dataObj` is declared so that the properties of the currently selected `ImageVO` instance can be retrieved and then saved to it (see Listing 8-60).



LISTING 8-60: The execute method for SaveImageCommand

```
public function execute(note:INotification):Void {
      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));

      var pImagePath:String = imgProxy.selectedLargeImagePath;
      var pTitle:String = imgProxy.selectedTitle;
      var pSelectedIndex:Number = imgProxy.selectedIndex;

      var dataObj:Object = { imageTitle: pTitle,
                             imagePath: pImagePath,
                             selectedIndex: pSelectedIndex };
}
```

The properties of the selected `ImageVO` instance in `dataObj` can be saved by calling the `saveSO()` method of a `SharedObjectProxy` instance (see Listing 8-61).



LISTING 8-61: The execute method for SaveImageCommand continued

```
public function execute(note:INotification):Void {

      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));

      var pImagePath:String = imgProxy.selectedLargeImagePath;
      var pTitle:String = imgProxy.selectedTitle;
      var pSelectedIndex:Number = imgProxy.selectedIndex;

      var dataObj:Object = { imageTitle: pTitle,
                             imagePath: pImagePath,
                             selectedIndex: pSelectedIndex };

      var soProxy:SharedObjectProxy;
      soProxy = SharedObjectProxy(facade.retrieveProxy(SharedObjectProxy.NAME));
      soProxy.saveSO(dataObj);

}
```

GetStoredImageCommand.as

In retrieving the data, `GetStoredImageCommand` also retrieves `SharedObjectProxy` and makes a call to the local shared object through calling the `init()` method of the proxy (see Listing 8-62).

www.it-ebooks.info


Available for download on Wrox.com

LISTING 8-62: The execute method for GetStoredImageCommand

```
public function execute(note:INotification):Void {
      var soProxy:SharedObjectProxy =
                  SharedObjectProxy(facade.retrieveProxy(SharedObjectProxy.NAME));
      soProxy.init("imageViewer");
}
```

Displaying the Image Grid

Next, take a look at the commands that are involved in displaying images in the image grid.

ViewImageGridCommand.as

A call to display the image grid is basically done through `ViewImageGridCommand`. The `VIEW_IMAGE_GRID` notification is sent and is then handled by `ImageGridMediator`. The images are sent in the notification body (see Listing 8-63).


Available for download on Wrox.com

LISTING 8-63: The execute method for ViewImageGridCommand

```
public function execute(note:INotification):Void {
      var images:Array = new Array(note.getBody());
      sendNotification(ApplicationFacade.VIEW_IMAGE_GRID, images);
}
```

SelectFirstImageCommand.as

`SelectFirstImageCommand` is called to set the first image that should be shown in the image grid. From the resulting `GET_STORED_IMAGE` notification, `ImageGridMediator` will make a call to `SelectFirstImageCommand` and will set the first image to be the last selected image in the image grid by passing the index value via the notification body. If no data is retrieved from the `SharedObject`, or the call to `GET_STORED_IMAGE` is unsuccessful, then the default first image index is automatically set to `0`. The `SELECTED_INDEX_UPDATED` notification is sent here also with the selected image in `ImageProxy` (see Listing 8-64).


Available for download on Wrox.com

LISTING 8-64: The execute method for SelectFirstImageCommand

```
public function execute(note:INotification):Void {
      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));

      var imgIndex:Number = Number(note.getBody());

      if(!imgIndex){
            imgProxy.selectedIndex = 0;
      } else {
            imgProxy.selectedIndex = imgIndex;
      }

     sendNotification(ApplicationFacade.SELECTED_INDEX_UPDATED,
                  imgProxy.selectedImage);
}
```

www.it-ebooks.info

Displaying a Single Image

When a user selects an image from the image grid, `ViewSingleImageCommand` is called. This is actually done through `SelectImageCommand`, which extends `MacroCommand`.

SelectImageCommand.as

`SelectImageCommand` sequentially executes `SaveImageCommand` and `ViewSingleImageCommand` (see Listing 8-65).



LISTING 8-65: The initializeMacroCommand method for SelectImageCommand

```
public function initializeMacroCommand():Void {
      addSubCommand(new SaveImageCommand());
      addSubCommand(new ViewSingleImageCommand());
}
```

ViewSingleImageCommand.as

`ViewSingleImageCommand` simply retrieves `SingleImageMediator` and calls the `loadImage()` method, passing in the `largePath` of `ImageVO`. Of course the image path is retrieved from the getter method `selectedLargeImagePath` called on the `ImageProxy` instance (see Listing 8-66).



LISTING 8-66: The execute method for ViewSingleImageCommand

```
public function execute(note:INotification):Void {
      var imgMediator:SingleImageMediator =
            SingleImageMediator(facade.retrieveMediator(SingleImageMediator.NAME));

      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));

      imgMediator.loadImage(imgProxy.selectedLargeImagePath);
}
```

SetIndexCommand.as

`SetIndexCommand.as` is used by `ImageGridMediator` to update `ImageProxy` (see Listing 8-67).



LISTING 8-67: The execute method for ViewSingleImageCommand

```
public function execute(note:INotification):Void {
      var imgProxy:ImageProxy = ImageProxy(facade.retrieveProxy(ImageProxy.NAME));
      imgProxy.selectedIndex = Number(note.getBody());
}
```

www.it-ebooks.info

Viewing the Application in Device Central

You have now covered each aspect of the image viewer client. In Chapter 13, you'll take a look at packaging this application for distribution.

If you haven't already, run the completed app in Device Central. Ensure that you have selected the Nokia E71 for testing. Figure 8-9 shows the completed grid view for the Image Viewer on the Nokia E71 in Device Central.



FIGURE 8-9

SUMMARY

In this chapter you learned how to develop a simple Image Viewer client for Flash Lite using PureMVC.

You covered some of the key considerations for using images in Flash Lite applications, including supported images, MIME types, memory considerations, image sizing, and loading multiple images.

You explored the key classes behind displaying both a single image in the view and displaying multiple images in a custom image grid.

You were also introduced to the `SharedObject` class and learned how to store data on and retrieve data from the device.

In the next chapter you'll explore the world of Twitter and take a look at creating a Twitter Client using PureMVC.

www.it-ebooks.info

9

Creating a Twitter Client

WHAT'S IN THIS CHAPTER?

- Exploring the core Twitter Developer APIs and methods
- Using the SWX format for Twitter API calls
- Using PureMVC to build a Twitter client

Popular Web sites such as Twitter, YouTube, MySpace, and Facebook are used by millions of people daily and highlight the success of social networking on the Internet. In their own unique way, each of these sites offers a service that allows users to share a little bit of their own personality with the world. Whether it's sharing favorite video clips or simply sending text-based messages to friends and family, these sites offer an alternative method to communication on a huge scale.

In this chapter you'll take a look at developing your very own social networking application, a Twitter client, using aspects of the official Twitter API Wiki (`http://apiwiki.twitter.com/`), which details the URLs required to access Twitter, and also Aral Balkan's SWX PHP service (SWX, `http://www.swxformat.org`), a gateway to providing more simplistic data from the Twitter service.

Twitter is one of the success stories of Web 2.0 for both social networking and Internet communication. The online service allows users to post messages to other Twitter users known as followers, messages that consist of a specific number of characters, 140 characters to be exact. Twitter has a minimalist interface that consists of a single input text field for your latest update, a button for submitting your update, and a list of updates from people whom you follow. Along the way you'll also learn some of the terminology you'll come across in the world of Twitter.

www.it-ebooks.info

EXPLORING THE TWITTER API

In this section you'll get an overview of the Twitter API and the components you'll use to build a Twitter client for mobile, using the PureMVC framework. The Twitter site embraces feedback from the developer community and encourages developers to come up with their own innovative variations and mashups of the Twitter service, so at the end of this chapter you should have many ideas on how you can build innovative mobile applications using the service. Some very successful applications have already been developed, using the API, for browser, desktop, and mobile clients.

The Twitter Developer API Wiki provides a comprehensive set of APIs that are open for Flash developers to build applications with. They consist of the same functions as featured on the live Twitter pages that are updated by millions of Twitter users around the world, every day. You can actually mirror the entire feature set of the Twitter site using Flash as the front end, if you really want to.

Twitter Fundamentals

Before delving into the detail of the Twitter API, it makes sense to go over some of the Twitter fundamentals.

Creating a Twitter User Account

Twitter is a free service, and creating a Twitter user account is relatively simple. You can sign up for an account on the Twitter Website at `https://twitter.com/signup?commit=Join!`.

You'll need a Twitter user account to access some portions of the API. After joining Twitter, or if you have joined already, at various points in the chapter you will need to substitute your login credentials in some of the examples. For example, the `verifyCredentials` call referenced in the SWX Twitter API, which you will cover later, will require you to supply your Twitter ***username*** and ***password*** for authentication. The following SWX AS call demonstrates our example:

```
swx.Twitter.verifyCredentials(["username ", "password"], resultHandler));
```

You'll cover the Twitter `verifyCredentials` call in more detail later.



You should never embed your username and password into your. `fla` *files or ActionScript code, because it is not secure. A safer option is to include them in server-side scripting, but again this isn't entirely secure.*

Twitter Phrases and Terminology

Following is a list of common phrases to guide you through understanding some of the concepts and features you're going to include in your Twitter client:

- Twitter status
- Tweets
- Retweeting

www.it-ebooks.info

- Followers, following, and blocking
- Replies and mentions
- Timelines
- Direct messaging
- HashTags and trends

The Twitter Status

The "Twitter status" is a user's most recent Twitter update. It simply represents the last message posted by the user to his or her Twitter page, consisting of no more than 140 characters of text. The Twitter client that you explore in this chapter will be able to update a Twitter status for a single user. Posting more characters truncates the update and splits it into a division of 140. Each Twitter status has a time stamp indicating when the status was last updated by the user.

Tweets

The term "tweets" represents the status updates a user has made to his or her Twitter page; it can simply refer to the total number of updates or the collection of updates.

Retweeting

A "retweet" is where a Twitter user reproduces a status update, usually word for word. The update is usually preceded by the letters "RT" in the new post.

Following and Blocking

Twitter allows you to "follow" other Twitter users who have public profiles. When you follow another Twitter user, you add yourself to that user's collection of followers. The total number of users *following* and the total number of *followers* is tracked and displayed in every Twitter user's status page. You can stop Twitter users from following you by blocking them. This will also remove them from your list of followers.

Timelines

Timelines are views displaying a chronological order of status updates from various Twitter users. There are currently three timelines: A user's timeline displays his or her own personal status collection, the public timeline contains updates of all public Twitter users, and the friends' timeline is a timeline representing the statuses of those you are following.

HashTags and Trends

A "hashTag" is like a category. A user adding a hash (#) symbol before a word in his or her status creates a hashTag. For the status "Jai wants to be a #JEDI like #Anakin," "#JEDI," and "#Anakin" are the hashTags, turning the words "JEDI" and "Anakin" into categorized keywords that can be used in Twitter searches and trend information.

Direct Messaging

Twitter allows you to send private messages directly to other Twitter users that don't appear on any timeline.

www.it-ebooks.info

Replies and Mentions

Replies are a Twitter user's responses to a tweet posted by another Twitter user. It can be seen anywhere along a user's timeline and doesn't have to be the latest post. Replies are typically started with the at (@) symbol, followed by the actual text of the reply.

"Mentions" are a Twitter user's reference to another Twitter user in a status update, which is not in reply to an existing post — for example, "Currently at the cinema with @jganderson."

Each of the terms mentioned here provides us with some concepts to include in our Twitter client.

Twitter API Methods

Now that you have a good understanding of the common phrases and terminology used in Twitter, let's take a look at the Twitter API methods currently available.

The Twitter API is effectively broken down into two main types, Search and REST. These have slightly different return structures, which makes understanding them a little confusing at first.

Our Twitter client will use the REST methods, and not the Search portion of the API. Each REST method's response can be retrieved in a particular format from XML, JSON, RSS, or Atom. The API documents the URL for each call independently, giving you the simple choice as to which format to use.

Each Twitter API call made over HTTP varies and will require either a `GET` or `POST` as the HTTP method. Generally, updates and methods requiring authentication require the `POST` HTTP method, whereas destroy methods have `DELETE` HTTP methods.

The typical structure of a `REST` HTTP request looks like this:

```
http://twitter.com/<methodCategory>/<methodName>.format
```

There are two exceptions to the structure. When the method requires an `id` to be referenced in the HTTP request, then the structure would be something like this:

```
http://twitter.com/<methodCategory>/<methodName>/id.format
```

Following this example for each type of API call, you will need to substitute both `<methodCategory>` and `<methodName>`.

The `<methodCategory>` can be substituted for one of the following values:

- `account`: for methods associated with a Twitter user's credentials
- `blocks`: for methods associated with blocking Twitter users
- `direct_messages`: for methods associated with sending private messages straight to another Twitter user
- `friendships`: for methods associated with managing Twitter friendships
- `favorites`: for methods associated with assigning favorite Twitter posts

www.it-ebooks.info

- `friends`: for methods associated with viewing Twitter friends
- `statuses`: for methods associated with Twitter status

As more features are added to Twitter, you can be sure the number of API methods available is likely to grow. Next let's take a look at the `<methodName>` for each category and call some examples along the way.

Account Requests

The basic structure for an XML-based `account` request over HTTP looks like this:

```
http://twitter.com/account/<methodName>.xml
```

Simply replace `<methodName>` with one of the following calls:

- `verify_credentials`: To verify that a username and password are valid
- `end_session`: To end a Twitter session
- `rate_limit_status`: To view the remaining number of API requests a user can make before the rate limit is reached for the current hour
- `update_delivery_device`: To update the delivery device
- `update_profile_colors`: To update the profile colors
- `update_profile_image`: To update the profile image
- `update_profile_background_image`: To update the background image of the user's Twitter page
- `update_profile`: To update the name, URL, location, and description on a Twitter user's profile

The style method settings listed here — `update_profile_colors`, `update_profile_image`, and `update_profile_background_image` — could be used to style the Twitter client mobile application. This is not something that is included but gives you an idea of what you could potentially use as part of the design. For each call, you also need to ensure that you have the correct parameters supplied.

Verifying User Credentials

Both `verify_credentials` and `end_session` API methods require authentication. Use the following steps to verify your user credentials in the browser.

1. Open your browser window and enter the `verify_credentials` call (`http://twitter.com/account/verify_credentials.xml`) in the address bar.
2. In the dialog box that opens, enter your Twitter `<username>` and `<password>`.

   If authentication is successful, you should see the results in the browser window. You may need to right-click ➪ View Source to see the results in XML format. If authentication fails, you'll see an error message.

www.it-ebooks.info

In our Twitter client you replicate the authentication and login process using Flash Lite and the SWX Twitter API. This will allow a user to sign you in with a click of a button. If you are successfully authenticated, access to other APIs requiring authentication will automatically return a response if the parameters are correct. This is because the session for the authentication is still running. To end a session you need to call the `end_session` method to effectively sign out of the Twitter API.

Block Request

The basic structure for an XML-based `block` request over HTTP looks like this:

```
http://twitter.com/block/<methodName>.xml
```

where generally you can replace `<methodName>` with one of the following calls:

- `create`: to block another Twitter user from following your Twitter status updates
- `destroy`: to remove a block you have added to a user
- `exists`: to see if you are blocking a user
- `blocking`: to view a list of the Twitter users you are blocking

When using the `blocking` call, you can also specify a particular `id`:

```
http://twitter.com/block/blocking/id.xml
```

Direct Messages Request

Here's the basic structure for a direct message request:

```
http://twitter.com/direct_messages/<methodName>.xml
```

When calling this method category, you can replace `<methodName>` with one of the following calls:

- `sent`: to retrieve the direct messages you have sent to other Twitter users
- `new`: to create a new direct message
- `destroy`: to destroy and remove a direct message from your account

```
http://twitter.com/direct_messages/sent.xml
```

Friendships Request

Here's the basic structure for a `friendships` request:

```
http://twitter.com/friendships/<methodName>.xml
```

When calling the method category, simply replace `<methodName>` with one of the following calls:

- `create`: to create a new friendship
- `destroy`: to destroy and remove a friendship

www.it-ebooks.info

- `exists`: to determine whether a friendship exists between Twitter users
- `show`: to show the friendships that a Twitter user has

Favorites Request

Here's the basic structure for a `favorites` request:

```
http://twitter.com/favorites/<methodName>.xml
```

When calling the method category, replace `<methodName>` with one of the following calls:

- `create`: to create a new favorite
- `destroy`: to destroy an existing favorite

You can use this method to create or destroy favorite Twitter statuses.

Statuses Request

Here's the basic structure for all requests based on statuses:

```
http://twitter.com/statuses/<methodName>.xml
```

Replace `<methodName>` with one of the following to action a specific call:

- `public_timeline`: to retrieve the public timeline
- `friends_timeline`: to retrieve the Twitter statuses of Twitter users you are following
- `friends`: to retrieve a list of the Twitter users you are following
- `followers`: to retrieve a list of the Twitter users who are following you
- `user_timeline`: to retrieve a timeline from an individual Twitter user
- `mentions`: to retrieve a list of statuses that mention your Twitter name
- `show`: to retrieve an individual Twitter status using the `id`
- `update`: to update your Twitter status
- `destroy`: to destroy and remove a Twitter status from your timeline

Whole chapters have been written on the methods and calls of the Twitter API. It's not my intention repeat that here; I recommend you check out the actual API Wiki, which is regularly updated.



At this point I really should mention rate limiting. The rate limit is the total number of calls a client can make to the Twitter API in a set amount of time. The Twitter API allows only 150 calls to be made to the REST API methods over an hour. When you use the public SWX gateway in the next section you may notice that some calls are not successful because the rate limit has been reached. By calling the `rate_limit_status` *API method, you can confirm whether the maximum has been reached.*

www.it-ebooks.info

THE SWX FORMAT

In this section you'll familiarize yourself with Aral Balkan's SWX format and learn how to use the SWX Twitter API service and methods to seamlessly integrate data straight into your Flash mobile applications.

For our Twitter client you'll use the Twitter APIs to:

- Verify user accounts
- Update the user statuses
- View friends' timelines

The main aim in this section is to understand how using the SWX format enables you to create online services that you can use as an intermediary for connecting with your Flash Lite applications. You'll then see how easy it is to use the Twitter API using the SWX format.

What Is SWX?

SWX is described as "The Native Data Format for Flash." It is essentially a data format comprising the SWF bytecode, containing raw data that can be used directly within a Flash application when it is loaded. It can be used without the need for explicit parsing, so variables, properties, objects, and their associated returned values can be referenced in your applications with ease.

As with the majority of open source and developer APIs, there are a few terms you may come across when reading aspects of SWX, including SWX RPC and SWX PHP, so let's clear these up.

Using SWX RPC

SWX RPC is the term that defines the remote procedure call encoded in the SWX format, used for lightweight data exchange, making it an ideal solution for mobile applications. Any request you make using SWX RPC enables you to retrieve data from pre-defined service-based classes, residing on back-end servers, and the results of each method and service call are returned in the SWX format, which can then be used directly in Flash mobile applications without the need for parsing, as with XML.

The introduction of the SWX was greeted with a sigh of relief from developers who had longed for an RPC protocol to be supported by Flash Lite mobile applications, enabling lightweight integration of data suited to mobile devices.

SWX RPC is open source and has been ported to various server-side technologies including Java, .NET, Ruby, and PHP.

Using SWX PHP

SWX PHP defines the implementation of SWX RPC on PHP servers. This means that all the back-end methods and classes defined to call services are written in PHP. This is the version used for the Twitter client.

www.it-ebooks.info

You can find version 2.0.0 beta of SWX PHP along with the code for the Twitter client. This will allow you to install SWX PHP on your local or remote Web servers, enabling you to try out the API services, which we'll cover next.

Download the SWX Package

You can also download the latest SWX PHP package from Google Code:

```
http://code.google.com/p/swx-format/
```

I recommend that you install a copy of SWX to your own Web server if you have one, or use the online service located at `http://swxformat.org`.

Creating a Service Call

The typical SWX RPC call requires you to provide the following parameters in ActionScript when making a request:

- `serviceClass`: name of the class representing the service
- `method`: name of the method you want to call in the class
- `args`: parameters specific to the method

These parameters essentially define the SWX service call.

If you navigate to the `/swx/services` folder in your SWX PHP download/installation directory you'll see a number of service classes, including the `Twitter.php` file. Open the file and you'll find all the methods that can be used to access the Twitter API.

Note that each service for SWX PHP is defined by the individual class. In our case this is the `Twitter` service class.

Table 9-1 lists each of the methods and required parameters that can be referenced from the SWX Twitter service class.

TABLE 9-1: SWX Twitter service API methods and parameters

METHOD NAME	PARAMETERS
`destroy`	`id, username, password`
`destroyDirectMessage`	`messageId, username, password`
`directMessages`	`username, password, since`
`endSession`	`username, password`
`followerFriends`	`friendId, friendusername, username, password`
`followers`	`username, password`
`followersWhoAreNotFriends`	`username, password`
`friends`	`username, password, twitterUser, page`

continues

www.it-ebooks.info

TABLE 9-1 *(continued)*

METHOD NAME	PARAMETERS
friendsNoAuth	twitterUser
friendsTimeline	username, password, [twitteruser], [since], [count], [page]
friendshipCreate	twitterUser, username, password
friendshipDestroy	twitterUser, username, password
isFriend	username, password, twitterUser(A), [twitteruser(B)]
leaveFriend	friendId, friendUsername, username, password
newDirectMessage	recipient, messageStr, username, password
notifications	state, friendId, friendUsername, username, password
publicTimeline	*No params required*
rateLimitStatus	username, password
replies	username, password, [page]
search	searchStr, [language], [RPP], [sinceId], [geoCode], [showUser]
sentDirectMessages	username, password, [page], [since], [sinceId]
showStatus	statusId
showUser	twitterUser, username, password
update	updateStr, username, password, [source], [replyId]
update_delivery_device	username, password, deviceSMS
update_location	username, password, location
userTimeline	username, password, id, [count], [since]
verifyCredentials	username, password

User Timeline

Let's take a look at `userTimeline`, one of the methods from Table 9-1. Once you've downloaded SWX PHP, open the `Twitter.php` file found in the SWX package and browse to the `userTimeline()` function. Although these functions are written in PHP, they are pretty straightforward to understand. To begin with, this method requires authentication, so when you

www.it-ebooks.info

call the method you need to supply your own username and password. These parameters are shown in the following snippet as the variables `$user` and `$pass`:




```
function userTimeline($user, $pass, $id = NULL, $count = 50, $since = NULL)
{
      $url = 'twitter.com/statuses/user_timeline.json';
      $since = (empty($since)) ? NULL: $since;
      $vars = array('id' => $id, 'count' => $count, 'since' => $since);
      $response = $this->_jsonCall($url, $vars, 'GET', $user, $pass);

      return $response;
}
```

code snippet Twitter.php

Each call in the `Twitter.php` class uses the `json` format of the Twitter API. The `userTimeline` call is referenced in the the `$url` reference:

```
$url = 'twitter.com/statuses/user_timeline.json';
```

What's also important to understand here are the arguments for the `userTimeline()` method calls `$user`, `$pass`, `$id`, `$count`, and `$since` need to be passed from the `userTimeline()` AS call to the SWX PHP service. Make note of the order of the parameters:




```
import org.swxformat.*;
var swx:SWX = new SWX();
swx.gateway = "http://swxformat.org/php/swx.php";
swx.encoding = "GET";

var callParams:Object =
{
      serviceClass: "Twitter",
      method: "userTimeline",
      args: ["<username>", "<password>", "<username>", 10, "2010-02-01"]
}

swx.call(callParams);
```

code snippet chapter9_twitterUserTimeline_1.fla

If you look at the PHP functions in `Twitter.php`, you'll see that some of the arguments are actually optional, and in Table 9-1 these are represented by the square brackets. For the `userTimeline` call these are `[count]` and `[since]`, representing the number of results, and the date at which to start searching for results. If you run this method, be sure to substitute `<username>` and `<password>` with your own username and password.

www.it-ebooks.info



When using SWX PHP, you need to reference the SWX gateway. The public SWX gateway is referenced as `http://www.swxformat.org/php/swx.php`. *This pretty much allows anybody to use the default SWX APIs, but you can change this to your own installation. Take into account that the rate limit for the public SWX gateway is likely to be reached faster with more people using the same or similar calls.*

You will notice that the `SWX` class has an `encoding` property that is required for the HTTP method request. This setting should be set in accordance with the API call.

The `callParams` is used to supply each of the Service call parameters, `serviceClass`, `method`, and `args`.

You have an option of using different syntax for calling SWX APIs in ActionScript. In the previous code snippet the `userTimeline` was demonstrated as an example call for SWX, whereby all the parameters for the method were provided in the `callParams` and then supplied to the `call()` method of the SWX instance. The alternative is to call the SWX Twitter API method directly, using the `Twitter` service class on the SWX object. Here's how the alternative call would look for the `userTimeline` call:




```
swx.Twitter.userTimeline(["<username>", "<password>", "<username>", 10,
"2010-02-01"], resultHandler, true);
```

code snippet chapter9_twitterUserTimeline_2.fla

Notice here that the first parameter to the call is an array of parameters, whereas the second is an additional call parameter called `resultHandler`. The `resultsHandler` is for designating a method to call when the results of the service call have been completed. This is covered in detail when you explore the Twitter client later. The third parameter is the `debug` property, which you can use to define whether the SWX calls should run in `debug` mode.

In addition to the `resultsHandler`, other event handlers can be configured on the `swx` instance, including `faultHandler`, `progressHandler`, and `timeoutHandler`. These should be self-explanatory. By creating a `faultHandler` method, you can handle potential faults with calls to the SWX service. For the `progressHandler` property, you can retrieve the progress of the SWX call and calculate the percentage loaded via the `bytesLoaded` and `bytesTotal` properties:




```
swx.progressHandler = progressHandler;
function progressHandler(event:Object) {
      trace("progressHandler: " + event.bytesLoaded + " of " + event.bytesTotal);
}
```

code snippet chapter9_twitterUserTimeline_2.fla

www.it-ebooks.info

The `timeoutHandler` works in conjunction with the `timeout` property of the `swx` instance. For the `timeout` property, you can set the time, in seconds, at which the `timeoutHandler` event should be triggered for a call when there is no response from the server:




```
swx.timeout = 10;
swx.timeoutHandler = timeoutHandler;
function timeoutHandler(event:Object){
      trace("Call timeout. Please try again.");
}
```

code snippet chapter9_twitterUserTimeline_2.fla

Now that you've looked at the code behind the SWX service methods in PHP and example ActionScript to call the service methods, next try calling the other API methods for the service and examine the results returned from them.

Using the SWX Service Explorer

The easiest way to learn SWX API Services without examining the PHP is by looking at some example usages via the SWX Service Explorer. The Explorer is the first of two SWX tools you'll become familiar with in this section. The tool contains access to the default SWX APIs found in the SWX PHP package, allowing you to experiment with calls to online services, including Flickr, Nabaztag, and of course Twitter.

If you've installed the SWX package download to either a remote or local server, you just need to navigate to the location of the Explorer in a Web browser.

To follow the next steps you can either use the Public SWX Services Explorer, `http://swxformat.org/php/explorer/`, or your installed version, `<swxInstallation>/php/explorer/`, where `<swxInstallation>` is your installation path for SWX PHP.

1. To explore the default SWX APIs, open the Public SWX Services Explorer in your Web browser.
2. In the Preferences window that opens, ensure that the Gateway Location field is set to `http://swxformat.org/php/amf.php`.
3. In the Service Class panel on the left you will see a list of Public SWX Service APIs that you can readily use in your applications, including the Public SWX Twitter API Service. Select Twitter from the list.
4. To the right of the Service Class section the Exploring Twitter panel appears, and here you should see a drop-down list containing all the service methods that can be used in the Twitter API.
5. Select the `publicTimeline()` method from the drop-down list.

   The `publicTimeline` method returns the most recent statuses from nonprotected users.
6. Press the `call` button at the bottom of the SWX Services Explorer window.

www.it-ebooks.info

7. Underneath the list of service methods appears a subpanel with five tabs: Results, Recordset view, Tree view, Trace, and Info. In the Results, Recordset, and Tree views you will be able to see the properties and values returned. A successful call to the `publicTimeline` method returns a result in an array, where each object represents one item in the result.

If you recall, the `userTimeline` call is an example that requires authentication, so you need to supply your Twitter login details when calling these methods in the user and password input fields. The `publicTimeline` call, on the other hand, is an example that doesn't require authentication.

Take some time to familiarize yourself with these panels; each can help you with the results returned from an SWX API. Try experimenting with the calls and attempt to create your own SWX services.

Using the SWX Data Analyzer

The other tool you can use while developing SWX-based applications is the SWX Data Analyzer. The tool essentially lets you see the results that are returned via SWX PHP in a Tree view, which is handy for examining calls using the service. The analyzer details faults, errors, timeouts, and results returned by service calls.

As with the SWX Explorer, you have the option of using the public version of the tool found here: `http://swxformat.org/php/analyzer/`.

Or you can use your own version: `<swxInstallation>/php/analyzer/`.

Both options are valid and neither has any dependency on API rate limits.

For Flash Lite applications that run in Device Central, the debug is rather buggy. Try publishing your SWF file to Flash Player 8, removing or commenting out basic compiler errors such as `fscommand` calls, before running the SWF in the SWX Data Analyzer. Also remember to turn on debugging for your SWX files; set the `debug` property on the `swx` instance to `true`. You must also remember to open the Analyzer before running the SWF.

In the next section you'll explore the build for the Twitter client, using the PHP implementation of the SWX RPC and PureMVC.

BUILDING THE APPLICATION

In this section you'll explore the code behind the Twitter client Console application.

You'll look at the following areas:

- The `.fla` file
- Defining ApplicationFacade
- Creating a Model for the Twitter client
- Creating the Twitter client's Controller
- Exploring the View's Mediators

www.it-ebooks.info

The .fla File

Before you move on to the core of the application, on the first actions frame in the timeline of the `chapter9_TwitterClient.fla` file, you'll see the following startup code:




```
import com.wrox.chapter9.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
app.startup(this);
```

code snippet chapter9_TwitterClient.fla.as

On the stage for the application is an animated cloud background that scrolls horizontally. In the Library panel for the application you'll see several assets: a `SoftKeys` component, a `PushButton` component, a `ListTwoRowWithIcon` component, and a `ListSingleRowWithIcon` component, each of which were covered in Chapter 4.

Defining ApplicationFacade

To define ApplicationFacade, follow along in this section.

Declaring the Notification Names

In `ApplicationFacade.as` you'll find each of the public static variable notification names that essentially define all the functions of our Twitter client (see Listing 9-1).



LISTING 9-1: Notification names for the Twitter client

```
public static var STARTUP:String = "startup";
public static var STARTUP_COMPLETE:String = "startupComplete";
public static var UPDATE_SOFT_KEYS:String = "updateSoftKeys";
public static var LOGIN:String = "login";
public static var LOGIN_SUCCESS:String = "loginSuccess";
public static var SHOW_USER:String = "showUser";
public static var SHOW_USER_SUCCESS:String = "showUserSuccess";
public static var SHOW_FRIENDS_TIMELINE:String = "showFriendsTimeline";
public static var SHOW_FRIENDS_TIMELINE_SUCCESS:String =
       "showFriendsTimelineSuccess";
public static var AUTHENTICATE_USER:String = "authenticateUser";
public static var DISPLAY_USER_LOGIN:String = "displayUserLogin";
public static var VERIFY_CREDENTIALS_SUCCESS:String = "verifyCredentialsSuccess";
public static var VERIFY_CREDENTIALS_FAILURE:String = "verifyCredentialsFailure";
public static var UPDATE_STATUS:String = "updateStatus"
public static var UPDATE_STATUS_SUCCESS:String = "updateStatusSuccess"
public static var SELECTED_INDEX_UPDATED:String = "selectdIndexUpdated";
public static var QUIT:String = "quit";
public static var RETRIEVE_USER_DETAILS:String = "retrieveUserDetails";
public static var SHARED_OBJECT_LOAD_SUCCESS:String = "sharedObjectLoadSuccess";
public static var SHARED_OBJECT_LOAD_FAILED:String = "sharedObjectLoadFailed";
public static var SHARED_OBJECT_SAVED:String = "sharedObjectSaved";
```

www.it-ebooks.info

Registering the Commands

The commands mapped to the notification names for our Twitter client are defined in `initializeController()`, which registers commands for the application (see Listing 9-2).



LISTING 9-2: The initializeController method for ApplicationFacade

```
public function initializeController():Void {
      super.initializeController();
      registerCommand(STARTUP, StartupCommand);
      registerCommand(SHOW_USER, ShowUserCommand);
      registerCommand(AUTHENTICATE_USER, AuthenticateUserCommand);
      registerCommand(RETRIEVE_USER_DETAILS, RetrieveUserDetailsCommand);
      registerCommand(SHOW_USER_TIMELINE, ShowUserTimelineCommand);
      registerCommand(SHOW_FRIENDS_TIMELINE, ShowFriendsTimelineCommand);
      registerCommand(UPDATE_STATUS, UpdateStatusCommand);
      registerCommand(QUIT, QuitCommand);
}
```

Here you'll see that the following notifications are registered with the Controller:

- STARTUP
- SHOW_USER
- AUTHENTICATE_USER
- RETRIEVE_USER_DETAILS
- SHOW_USER_TIMELINE
- UPDATE_STATUS
- QUIT

For now, you just need to be aware of each of the notification names you'll cover. Their usage and the logic behind them will be explained in more detail as you go through the model, view, and controller.

In the next section you'll explore the model.

Creating a Model for the Twitter Client

In this section you'll examine the model defined for the application (see Figure 9-1).

www.it-ebooks.info

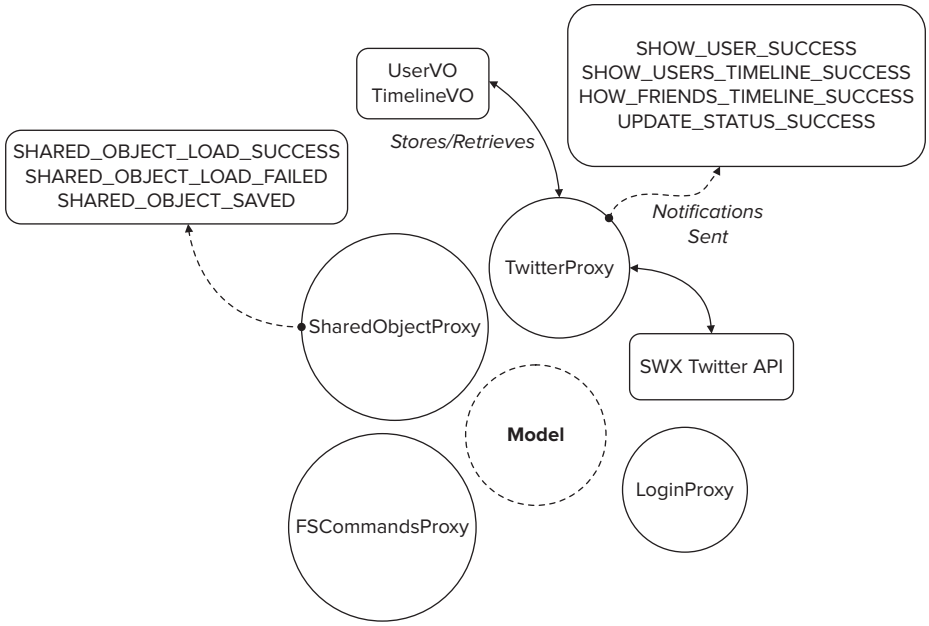


FIGURE 9-1

In the figure, you can see two new proxy classes that are used for the Twitter client application, `LoginProxy` and `TwitterProxy`. The `SharedObjectProxy` can also be seen. This was covered in Chapter 8, and the `FSCommandsProxy` shown in the figure was also covered in earlier chapters.

The `chapter9_TwitterClient.fla` file, accompanying this chapter, has the layout and library assets all set up ready for publishing from Flash CS4. The Project panel shown in Figure 9-2 indicates the file system layout for the model.

There are a number of notifications sent from the model which you'll take a look at later, but first take let's go through the value objects used in the application.



FIGURE 9-2

Defining the Value Objects

While SWX data can be used directly in Flash, logically our application will need to store the data in a clear and unambiguous way. There will be three value objects used in the application:

- `UserVO`: for general user details
- `ProfileStyleVO`: for user profile details
- `TimelineVO`: for a collection of updates relating to timelines

www.it-ebooks.info

UserVO.as

The `UserVO` class contains each of the properties associated with a user that can be retrieved from the Twitter API via SWX PHP, including `screenName`, `location`, `status`, `updates`, `friends`, `followers`, and the `profile` style instance (see Listing 9-3).



LISTING 9-3: The UserVO class

```
import com.wrox.chapter9.model.vo.ProfileStyleVO;

class com.wrox.chapter9.model.vo.UserVO {

      public var friends:String;
      public var followers:String;
      public var updates:String;
      public var status:String;
      public var id:String;
      public var name:String;
      public var screenName:String;
      public var location:String;
      public var description:String;
      public var profile:ProfileStyleVO;

      public function UserVO() {

      }

}
```

ProfileStyleVO.as

`ProfileStyleVO` is where the details of each user's profile design style settings are stored (see Listing 9-4). This class doesn't actually form a significant part of the completed Twitter client application, but it should give you an idea of the properties used that could be styled in the application.



LISTING 9-4: The ProfileStyleVO class

```
class com.wrox.chapter9.model.vo.ProfileStyleVO {

      public var textColor:Number;
      public var imageUrl:String;
      public var sidebarBorderColor:Number;
      public var sidebarFillColor:Number;
      public var backgroundColor:Number;
      public var backgroundTile:String;
      public var linkColor:Number;
      public var backgroundImageUrl:String;

      public function ProfileStyleVO() {

      }
}
```

www.it-ebooks.info

TimelineVO.as

`TimelineVO` simply contains the `updates` property, an `array`, which will contain a list of users when populated (see Listing 9-5).



LISTING 9-5: The TimelineVO class

```
class com.wrox.chapter9.model.vo.TimelineVO {

      public var updates:Array;

      public function TimelineVO(){

      }
}
```

Defining the Proxies

The model aspect of our Twitter client includes creating the following:

- `TwitterProxy`: to access the SWX Twitter API
- `LoginProxy`: to store and retrieve user credentials

Our Twitter app relies on the SWX gateway for retrieving user data using the Twitter API, so naturally you'll create `TwitterProxy` and use it to call the `statuses/friends_timeline`, `statues/users_timeline`, `statuses/update`, and `users/show` methods through SWX.

To access some of the API methods that require authentication, our app also needs to acquire the user's Twitter username and password. You'll need to capture these user details through a mediator and store them in `LoginProxy`.

`TwitterProxy` uses two parsers:

- `UserParser`: to parse the details of a user
- `TimelineParser`: to parse details of a timeline-based service call

UserParser.as

For `UserParser` the aim is to provide one avenue for saving the details of a Twitter user so they can be used at various aspects in the application. Each service call method in the Twitter SWX API will return a user's properties. The parser has one static method called `parseUser()` (see Listing 9-6).



LISTING 9-6: The UserParser class

```
import com.wrox.chapter9.model.vo.ProfileStyleVO;
import com.wrox.chapter9.model.vo.UserVO;

class com.wrox.chapter9.model.parsers.UserParser {
```

continues

www.it-ebooks.info

LISTING 9-6 *(continued)*

```
        public function UserParser() {

        }

        public static function parseUser(userObj:Object):UserVO {

                var userVO:UserVO = new UserVO();
                userVO.status = userObj.text;
                userVO.name = userObj.name;
                userVO.alias = userObj.screen_name;
                userVO.location = userObj.location;
                userVO.id = userObj.id;
                userVO.followers = userObj.followers_count;
                userVO.friends = userObj.friends_count;
                userVO.profile = new ProfileStyleVO();
                userVO.profile.backgroundColor = userObj.profile_image_url;
                userVO.profile.imageUrl = userObj.profile_image_url;

                return userVO;
        }
}
```

TimelineParser.as

In the application, the user will be able to choose to view a user timeline or a friend's timeline. The user should be able to view these at any point once they've been retrieved from the service without having to recall the SWX methods, `userTimeline` and `friendsTimeline`, when he or she changes views.

In `TwitterProxy` you therefore create two variables to reference `userTimeline` and `friendsTimeline`, and there are two possible timelines that a user will be able to select — the friend's timeline and his or her own user timeline. Similarly to other parsers we've covered, `TimelineParser` iterates through a number of results it retrieves through each service call and then returns an instance of `TimelineVO` with the updates array property populated with `UserVO` instances. This is then used in `TwitterProxy`. In the `parseTimeline()` method you'll notice that a new `TimelineVO` instance is created called `timelineVO`. The result of the `UserParser.parseUser()` method is the `updatesObj` is added to the updates property of the `timelineVO` value object (see Listing 9-7).



LISTING 9-7: The parseTimeline() method in TimelineParser

```
import com.wrox.chapter9.model.vo.TimelineVO;
import com.wrox.chapter9.model.parsers.UserParser;

class com.wrox.chapter9.model.parsers.TimelineParser {

        public function TimelineParser() {

        }
```

www.it-ebooks.info

```
        public static function parseTimeline(updatesObj:Object):TimelineVO {

              var timelineVO:TimelineVO = new TimelineVO;
              timelineVO.updates = [];

              for (var i:Number = 0; i < updatesObj.length; i++ ) {
                    timelineVO.updates.push(UserParser.parseUser(updatesObj[i]));
              }

              return timelineVO;
        }
  }
```

Now take a look at creating `TwitterProxy.as`.

Creating TwitterProxy.as

`TwitterProxy` will provide you with a single access point to the SWX gateway and Twitter API. The primary use of this proxy will be to initialize the SWX Twitter service. In this section you'll also create the public methods that use data retrieved through `TwitterProxy` for our application's needs.

For `TwitterProxy` you'll define the following core methods:

- `onRegister()`: to instantiate a new SWX instance
- `initCall()`: to initialize a call to the SWX Twitter API
- `resultsHandler()`: to handle results returned from the SWX Twitter API

Later in this section you'll also take a look at the following methods, which will be used by commands to forward data to our mediators:

- `getImagesFromFriendsTimeline()`: method to return a list of user profile images from a timeline
- `getScreenNamesFromFriendsTimeline()`: method to return a list of screen names from a timeline
- `getStatusesFromUserTimeline()`: method to return a list of statuses from a user's timeline
- `getUserAt()`: method to return a single Twitter user

For this walkthrough you'll need to open the empty `/wrox/chapter9/model/TwitterProxy.as` file.

1. Create our familiar Proxy class structure for our `TwitterProxy`, including the proxy `NAME` and public `onRegister()` method (see Listing 9-8).



LISTING 9-8: Creating TwitterProxy

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter9.model.TwitterProxy extends Proxy implements IProxy {
```

continues

www.it-ebooks.info

LISTING 9-8 *(continued)*

```
        public static var NAME:String = "TwitterProxy";

        public function TwitterProxy(data:Object) {
              super(NAME, data);
        }

        private function onRegister():Void {}

}
```

2. Import the `ApplicationFacade`, `Delegate`, `SWX`, `TimelineVO`, and `UserVO` classes (see Listing 9-9).



LISTING 9-9: Importing the core classes into Twitter Proxy

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;
import org.swxformat.SWX;
import com.wrox.chapter9.ApplicationFacade;
import com.wrox.chapter9.model.vo.UserVO;
import com.wrox.chapter9.model.vo.TimelineVO;
import mx.utils.Delegate;

class com.wrox.chapter9.model.TwitterProxy extends Proxy implements IProxy {

        public static var NAME:String = "TwitterProxy";

        public function TwitterProxy(data:Object) {
              super(NAME, data);
        }

        private function onRegister():Void {}

}
```

3. Declare a private variable for `SWX`, then in `onRegister()` create an instance of `SWX` called `swx` and define settings for the `gateway`, `encoding`, `timeout` seconds, `timeoutHandler`, `faultHandler`, and `resultHandler` properties (see Listing 9-10).



LISTING 9-10: Defining the SWX TwitterProxy

```
class com.wrox.chapter9.model.TwitterProxy extends Proxy implements IProxy {

        public static var NAME:String = "TwitterProxy";

        private var swx:SWX;

        public function TwitterProxy(data:Object) {
              super(NAME, data);
        }
```

www.it-ebooks.info

```
        private function onRegister():Void {
              swx = new SWX();
              swx.gateway = " ";s
              swx.encoding = "GET";
              swx.timeout = 10;
              swx.debug = true;

              swx.resultHandler = Delegate.create(this, resultHandler);
              swx.timeoutHandler = Delegate.create(this, timeoutHandler);
              swx.faultHandler = Delegate.create(this, faultHandler);
        }
}
```

4. Declare three private variables called `callParameters`, `callMethod`, and `callService` (see Listing 9-11).



LISTING 9-11: The SWX call arguments for the TwitterProxy

```
public static var NAME:String = "TwitterProxy";

private var callParameters:Object;
private var callMethod:String;
private var callService:String;
private var swx:SWX;

public function TwitterProxy(data:Object) {
      super(NAME, data);
}
```

5. Create the `initCall()` method. The method should take two arguments: `method`, a string representing an SWX Twitter method call, and `params`, an array of parameters that will be defined in a command call to `TwitterProxy`. These parameters are specific to the service `method`, which were covered earlier (see Listing 9-12).



LISTING 9-12: The initCall() method for TwitterProxy

```
public function initCall(method:String, params:Array):Void {
      callService = "Twitter";
      callMethod = method;
      callParameters = { serviceClass:callService,
                         method:callMethod,
                         args:params };

      swx.call(callParameters);
}
```

www.it-ebooks.info

Here you should notice that the `initCall()` method initiates our `SWX` service through the `swx.call()` method. The `callParameters` variable is an object that contains the properties `serviceClass`, `method`, and `args`. In `initCall()` you set the `callService` variable to "Twitter" and then assign the variable to the `serviceClass` property. The `callMethod` variable is defined from the first of the arguments of `initCall()` and the assigned to `method` property of `callParameters`. Then `params` is assigned to the `args` property of `callParameters`.

6. Next declare the public static class variables for each of the Twitter SWX method calls the application will use: `FRIENDS_TIMELINE`, `UPDATE`, `USERS_TIMELINE`, `SHOW_USER`, and `VERIFY_CREDENTIALS` (see Listing 9-13).



LISTING 9-13: Declaring static variables for SWX method calls in TwitterProxy

```
public static var NAME:String = "TwitterProxy";

public static var FRIENDS_TIMELINE:String = "friendsTimeline";
public static var UPDATE:String = "update";
public static var USERS_TIMELINE:String = "usersTimeline";
public static var SHOW_USER:String = "showUser";
public static var VERIFY_CREDENTIALS:String = "verifyCredentials";

private var callParameters:Object;
private var callMethod:String;
private var callService:String;private var swx:SWX;
```

These variables should be used to define the `method` argument when calling `initCall()`. You will also use these in the `resultHandler()` method, which you'll define next.

7. Create the `resultHandler()` method. The `resultHandler()` will return the `event` object, mentioned earlier, and because the `resultHandler()` method will be used for all the calls to the Twitter API service, you simply need to define a switch statement that handles each of the Twitter SWX method calls individually. Here the `callMethod` variable is supplied to the switch statement, whereby each `case` is one of the static variables defined in step 6 (see Listing 9-14).



LISTING 9-14: The resultHandler() method for TwitterProxy

```
private function resultHandler(event:Object):Void {
      switch(callMethod) {
            case SHOW_USER:
            break;
            case USERS_TIMELINE:
            break;
            case FRIENDS_TIMELINE:
            break;
            case UPDATE:
            break;
```

www.it-ebooks.info

```
            case VERIFY_CREDENTIALS:
            break;
      }
}
```

8. Import the `UserParser` and `TimelineParser` classes, then declare three more private class variables `twitterUser` with a type of `UserVO`, and then `friendsTimeline` and `userTimeline` each with data type of `TimelineVO` (see Listing 9-15).



LISTING 9-15: Importing parsers into TwitterProxy

```
import com.wrox.chapter9.ApplicationFacade;
import com.wrox.chapter9.model.parsers.UserParser;
import com.wrox.chapter9.model.parsers.TimelineParser;
import com.wrox.chapter9.model.vo.UserVO;
import com.wrox.chapter9.model.vo.TimelineVO;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;
import org.swxformat.SWX;

class com.wrox.chapter9.model.TwitterProxy extends Proxy implements IProxy {

      public static var NAME:String = "TwitterProxy";

      public static var FRIENDS_TIMELINE:String = "friendsTimeline";
      public static var UPDATE:String = "update";
      public static var USERS_TIMELINE:String = "usersTimeline";
      public static var SHOW_USER:String = "showUser";

      private var callParameters:Object;
      private var callMethod:String;
      private var callService:String;
      private var friendsTimeline:TimelineVO;
      private var swx:SWX;
      private var twitterUser:UserVO;
      private var userTimeline:TimelineVO;
```

9. Complete the `SHOW_USER` portion of the switch statement for `resultHandler()` by instantiating `twitterUser` and then using the static method `UserParser.parseUser()` to parse the result object `resultObj` to `twitterUser`. Send the `SHOW_USERS_SUCCESS` notification after parsing the user data (see Listing 9-16).



LISTING 9-16: Handling the SHOW_USER call in the resultHandler method for TwitterProxy

```
private function resultHandler(event:Object):Void {
      var resultsObj:Object = event.result;

      switch(callMethod) {
            case SHOW_USER:
```

continues

www.it-ebooks.info

LISTING 9-16 *(continued)*

```
                twitterUser = new UserVO();
                twitterUser = UserParser.parseUser(resultsObj);
                sendNotification(ApplicationFacade.SHOW_USER_SUCCESS);
            break;
            case USERS_TIMELINE:
            break;
            case FRIENDS_TIMELINE:
            break;
            case UPDATE:
            break;
            case VERIFY_CREDENTIALS:
            break;
        }
}
```

Here take note that before the switch statement, the result returned via the `resultHandler()`, `event.result`, is saved to the `resultObj` variable. This is also used in the other case statements.

10. Complete the `USERS_TIMELINE` and `FRIENDS_TIMELINE`. These cases are almost identical. Each call needs to use the `TimelineParser`. The only differences are that with `USERS_TIMELINE` you need to parse `resultsObj` to the `userTimeline` variable, and then send the `SHOW_USERS_TIMELINE_SUCCESS` notification, whereas with the `FRIENDS_TIMELINE` call, you need to parse `resultsObj` to the `friendsTimeline` variable and then send the `SHOW_FRIENDS_TIMELINE_SUCCESS` notification (see Listing 9-17).



LISTING 9-17: Handling the USERS_TIMELINE and FRIENDS_TIMELINE calls in the resultHandler method for TwitterProxy

```
switch(callMethod) {
      case SHOW_USER:
            twitterUser = new UserVO();
            twitterUser = UserParser.parseUser(resultsObj);
            sendNotification(ApplicationFacade.SHOW_USER_SUCCESS,
                                        twitterUser, "UserVO");
      break;
      case USER_TIMELINE:
            usersTimeline = new TimelineVO();
            usersTimeline = TimelineParser.parseTimeline(resultsObj);
            sendNotification(ApplicationFacade.SHOW_USER_TIMELINE_SUCCESS,
                                                  getStatusesFromUserTimeline());
      break;
      case FRIENDS_TIMELINE:
            friendsTimeline = new TimelineVO();
            friendsTimeline = TimelineParser.parseTimeline(resultsObj);
            var profileImages:Array = [];
            profileImages = getImagesFromFriendsTimeline();
```

www.it-ebooks.info

```
                var screenNames:Array = [];
                screenNames = getScreenNamesFromFriendsTimeline();
                var statusUpdates:Array = [];
                statusUpdates = getStatusesFromFriendsTimeline();
                sendNotification(ApplicationFacade.SHOW_FRIENDS_TIMELINE_SUCCESS,
                                                [profileImages, screenNames, statusUpdates]);
            break;
            case UPDATE:
            break;
            case VERIFY_CREDENTIALS:
            break;
    }
```

11. Complete `resultHandler()` by filling in the `UPDATE` portion of the switch statement. Here you simply send the `UPDATE_STATUS_SUCCESS` notification, along with the "Update Successful!" text in the notification body (see Listing 9.18).



LISTING 9-18: Handling the UPDATE call in the resultHandler method for TwitterProxy

```
switch(callMethod) {
      case SHOW_USER:
            twitterUser = new UserVO();
            twitterUser = UserParser.parseUser(resultsObj);
            sendNotification(ApplicationFacade.SHOW_USER_SUCCESS);
      break;
      case USERS_TIMELINE:
            userTimeline = new TimelineVO();
            userTimeline = parseTimeline(resultsObj);
            sendNotification(ApplicationFacade.SHOW_USERS_TIMELINE_SUCCESS);
      break;
      case FRIENDS_TIMELINE:
            friendsTimeline = new TimelineVO();
            friendsTimeline = TimelineParser.parseTimeline(resultsObj);
            var profileImages:Array = [];
            profileImages = getImagesFromFriendsTimeline();
            var screenNames:Array = [];
            screenNames = getScreenNamesFromFriendsTimeline();
            var statusUpdates:Array = [];
            statusUpdates = getStatusesFromFriendsTimeline();
            sendNotification(ApplicationFacade.SHOW_FRIENDS_TIMELINE_SUCCESS,
                                          [profileImages, screenNames, statusUpdates]);
      break;
      case UPDATE:
            sendNotification(ApplicationFacade.UPDATE_STATUS_SUCCESS,
                                                      "Update Successful!");
      break;
      case VERIFY_CREDENTIALS:
      break;
}
```

www.it-ebooks.info

12. Handle the `VERIFY_CREDENTIALS` call. Here you need to send the `VERIFY_CREDENTIALS_SUCCESS` notification along with the "Login Successful" text (see Listing 9-19).



LISTING 9-19: Handling the VERIFY_CREDENTIALS call in the resultHandler method for TwitterProxy

```
switch(callMethod) {
      case SHOW_USER:
            twitterUser = new UserVO();
            twitterUser = UserParser.parseUser(resultsObj);
            sendNotification(ApplicationFacade.SHOW_USER_SUCCESS,
                                          twitterUser, "UserVO");
      break;
      case USER_TIMELINE:
            usersTimeline = new TimelineVO();
            usersTimeline = TimelineParser.parseTimeline(resultsObj);
            sendNotification(ApplicationFacade.SHOW_USER_TIMELINE_SUCCESS,
                                                  getStatusesFromUserTimeline());
      break;
      case FRIENDS_TIMELINE:
            friendsTimeline = new TimelineVO();
            friendsTimeline = TimelineParser.parseTimeline(resultsObj);
            var profileImages:Array = [];
            profileImages = getImagesFromFriendsTimeline();
            var screenNames:Array = [];
            screenNames = getScreenNamesFromFriendsTimeline();
            var statusUpdates:Array = [];
            statusUpdates = getStatusesFromFriendsTimeline();
            sendNotification(ApplicationFacade.SHOW_FRIENDS_TIMELINE_SUCCESS,
                                    [profileImages, screenNames, statusUpdates]);
      break;
      case UPDATE:
            sendNotification(ApplicationFacade.UPDATE_STATUS_SUCCESS,
                                                            "UpdateSuccessful!");
      break;
      case VERIFY_CREDENTIALS:
            twitterUser = new UserVO();
            twitterUser = UserParser.parseUser(resultsObj);
            sendNotification(ApplicationFacade.VERIFY_CREDENTIALS_SUCCESS,
                                                            "Login Successful!");
      break;
}
```

13. Create the `faultHandler()` and `timeoutHandler()` methods. Here, just handle the `VERIFY_CREDENTIALS` request. The `faultObj.error` property will return the text "Could not authenticate you." when the `faultHandler` is invoked. The aim here is to display this fault to the user, so this value is altered to "Wrong Username/Email and password."

www.it-ebooks.info

The VERIFY_CREDENTIALS_FAILURE notification is sent with the message. For the timeoutHandler() the string "Request Timeout, please try again . . ." is defined and is also sent in the body of the VERIFY_CREDENTIALS_FAILURE notification (see Listing 9-20).



LISTING 9-20: The faultHandler and timeoutHandler methods for TwitterProxy

```
private function faultHandler(event:Object):Void {
      var faultObj:Object = event.fault;
      switch(callMethod) {
            case VERIFY_CREDENTIALS:
                  if (faultObj.error == "Could not authenticate you.") {
                        faultObj.error = "Wrong Username/Email and password.";
                  }
                   sendNotification(ApplicationFacade.VERIFY_CREDENTIALS_FAILURE,
faultObj.error);
            break;
      }
}

private function timeoutHandler(event:Object):Void {
      var timeoutStr:String = "Request Timeout, please try again..";
      switch(callMethod) {
            case VERIFY_CREDENTIALS:
     sendNotification(ApplicationFacade.VERIFY_CREDENTIALS_FAILURE, timeoutStr);
            break;
      }
}
```

Should you get to the point where you want to add functionality to the Twitter client, you could add more cases for the fault and timeout handlers.

14. Create the getImagesFromFriendsTimeline() method. The aim here is to return an array of URL strings pointing to each user's profile image. These are used in UserStatusMediator (see Listing 9-21).



LISTING 9-21: The getImagesFromFriendsTimeline() method for TwitterProxy

```
public function getImagesFromFriendsTimeline():Array {
      var images:Array = [];
      for(var i:Number = 0; i < friendsTimeline.updates.length){
            var userVO:UserVO = friendsTimeline.updates[i];
            images.push(userVO.profile.imageUrl);
      }
      return images;
}
```

www.it-ebooks.info

15. Create the `getScreenNamesFromFriendsTimeline()` method. This should return an array of screen names from Twitter users (see Listing 9-22).



LISTING 9-22: The getScreenNamesFromFriendsTimeline() method for TwitterProxy

```
public function getScreenNamesFromFriendsTimeline():Array {
      var screenNames:Array = [];
      for(var i:Number = 0; i < friendsTimeline.updates.length){
            var userVO:UserVO = friendsTimeline.updates[i];
            screenNames.push(userVO.screenName);
      }
      return screenNames;
}
```

16. Specify the `getUserAt()` method to return the user at a selected `index` (see Listing 9-23).



LISTING 9-23: The getUserAt method for TwitterProxy

```
public function getUserAt(index:Number):UserVO {
      var user:UserVO = friendsTimeline.updates[index];
      return user;
}
```

17. Specify the `getStatusesFromUserTimeline()` method. Here you are returning an array of tweet posts for the user (see Listing 9-24).



LISTING 9-24: The getStatusesFromUserTimeline() method for TwitterProxy

```
public function getStatusesFromUserTimeline():Array {
      var tweets:Array = [];
      for(var i:0; i < friendsTimeline.updates.length){
            var user:UserVO = friendsTimeline.updates[i];
            tweets.push(user.status);
      }
      return tweets;
}
```

18. Create the `GET` method `screenName` for returning the current user's screen name (see Listing 9-25).



LISTING 9-25: The screenName method for TwitterProxy

```
public function get screenName():String {
      return twitterUser.screenName;
}
```

www.it-ebooks.info

This wraps up `TwitterProxy`. Obviously we could do a whole lot more with `TwitterProxy`, but for the purpose of this application you're just trying to gain an understanding of how to use certain aspects of the data.

The completed `TwitterProxy` can be found in the .zip file accompanying this chapter.

In the next section you'll see how these methods are used in the application, focusing on the controller and our commands. First, take a look at `LoginProxy`.

LoginProxy.as

`LoginProxy` is simply a class of getters and setters to save and retrieve references to the authenticating user's username and password (see Listing 9-26). Various commands can then access the proxy and retrieve the values to action calls to the Twitter API that require authentication.



LISTING 9-26: The LoginProxy for Twitter client

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter9.model.LoginProxy extends Proxy implements IProxy {

      public static var NAME:String = "LoginProxy";

      private var _username:String;
      private var _password:String;

      public function LoginProxy(data:Object) {
            super(NAME, data);
      }

      public function set username(u:String):Void {
            _username = u;
      }

      public function get username():String {
            return _username;
      }

      public function set password(p:String):Void {
            _password = p;
      }

      public function get password():String {
            return _password;
      }
}
```

The Twitter client also uses `SharedObjectProxy` to save these details, which we'll discuss shortly.

www.it-ebooks.info

Creating the Twitter Client's Controller

In this section you'll examine the model defined for the application (see Figure 9-3).



FIGURE 9-3

In Figure 9-3, you can see seven new commands which are used in the Twitter client application, `VerifyCredentialsCommand`, `ShowUserCommand`, `ShowFriendsTimelineCommand`, `RetrieveUserDetailsCommand`, `StoreLoginDetailsCommand`, `UpdateStatusCommand`, and `AuthenticateUserCommand`. `StartupCommand`, `ViewPrepCommand`, `ModelPrepCommand`, and `QuitCommand` can also be seen; each of these has been covered in earlier chapters.

The Project panel shown in Figure 9-4 indicates the file system layout for the Controller.

www.it-ebooks.info

In this section you'll examine each of the new commands created for our Twitter client. These commands will allow our application to do the following:

- Prepare the application
- Handle authentication to a user's Twitter account
- Capture, store, and retrieve user login details
- Show Twitter timelines
- Update Twitter profiles and statuses

The Controller will predominantly consist of command calls using `TwitterProxy` and information retrieved from SWX PHP to send to the rest of the app. Our `LoginProxy` will also be used by the application for authentication purposes.



FIGURE 9-4

Preparing the Application

The entry point to the application uses our familiar `StartupCommand` (see Listing 9-25).

StartupCommand.as

`StartupCommand` consists of executing `ModelPrepCommand` and `ViewPrepCommand` sequentially (see Listing 9-27).

LISTING 9-27: The initializeMacroCommand method for StartupCommand

Available for download on Wrox.com

```
public function initializeMacroCommand():Void {
        addSubCommand(new ModelPrepCommand());
        addSubCommand(new ViewPrepCommand());
}
```

ModelPrepCommand.as

`ModelPrepCommand` registers `FSCommandsProxy`, `TwitterProxy`, `LoginProxy`, and `SharedObjectProxy` (see Listing 9-28).

LISTING 9-28: The execute method for ModelPrepCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {
        facade.registerProxy(new FSCommandsProxy());
        facade.registerProxy(new TwitterProxy());
        facade.registerProxy(new LoginProxy());
        facade.registerProxy(new SharedObjectProxy());
}
```

www.it-ebooks.info

ViewPrepCommand.as

`ViewPrepCommand` registers `UserLoginMediator`, `UserStatusMediator`, `UserTimelineMediator`, `FriendsTimelineMediator`, and `SoftKeyMediator`. The `DISPLAY_USER_LOGIN` notification is then sent once each of the mediators has been registered (see Listing 9.29).



LISTING 9-29: The execute method for ViewPrepCommand

```
public function execute(note:INotification):Void {
      var mc:MovieClip = MovieClip(note.getBody());
      facade.registerMediator(new UserLoginMediator(mc));
      facade.registerMediator(new UserStatusMediator(mc));
      facade.registerMediator(new UserTimelineMediator(mc));
      facade.registerMediator(new FriendsTimelineMediator(mc));
      facade.registerMediator(new SoftKeyMediator(mc));

      sendNotification(ApplicationFacade.DISPLAY_USER_LOGIN);

}
```

Handling Authentication

As mentioned previously, the application will need to capture a user's username and password, to allow that user to access the majority of the features of the Twitter client. Requesting information in our application will mean that the login details supplied need to be correct.

The commands involved in the authentication processes include:

- `AuthenticateUserCommand`: called when the user wants to log in to his or her Twitter account
- `VerifyCredentialsCommand`: called to verify a user's login details
- `StoreLoginDetailsCommand`: called to save a user's login details
- `RetrieveUserDetailsCommand`: called to retrieve a user's saved login details, if they exist

AuthenticateUserCommand.as

`AuthenticateUserCommand` extends `MacroCommand` and effectively initializes the `execute()` methods of `VerifyCredentialsCommand` and `StoreLoginDetailsCommand` sequentially (see Listing 9-30).



LISTING 9-30: AuthenticateUserCommand for the TwitterClient

```
import com.wrox.chapter9.controller.StoreLoginDetailsCommand;
import com.wrox.chapter9.controller.VerifyCredentialsCommand;
import org.puremvc.as2.patterns.command.MacroCommand;

class com.wrox.chapter9.controller.AuthenticateUserCommand extends MacroCommand {
      public function initializeMacroCommand():Void {
```

www.it-ebooks.info

```
            addSubCommand(new VerifyCredentialsCommand());
            addSubCommand(new StoreLoginDetailsCommand());
        }
    }
```



For every command that uses the SWX Twitter API, via `TwitterProxy`*, the* `retrieveProxy()` *method is used, then the* `initCall()` *method on the* `TwitterProxy` *instance is called, supplying one of the public static method name variables defined in* `TwitterProxy`*. For instance, to call the SWX Twitter API* `showUser` *function, you would supply* `Twitter.SHOW_USER` *as the first argument for* `initCall()`*. You'll see examples of this in each command.*

In addition to `VerifyCredentialsCommand`, several other commands will be used in the application for accessing the SWX Twitter API, including:

- `VerifyCredentialsCommand`: to verify that the user's username and password are valid
- `ShowUserCommand`: to show the user's latest status
- `ShowFriendsTimelineCommand`: to show the friends timeline
- `ShowUserTimelineCommand`: to show the user's timeline
- `UpdateStatusCommand`: to update a user's status

Each of the commands imports the `TwitterProxy` you created earlier.

Capturing a User's Login Details

`VerifyCredentialsCommand` is where the user's username and password entered in the application are passed to `TwitterProxy` for authentication.

VerifyCredentialsCommand.as

In the `execute()` method for `VerifyCredentialsCommand`, notice that the `note.getBody()` method is an array whose values are supplied as the second argument of `initCall()`, which you may remember represents our parameters that are sent to the SWX Twitter API call (see Listing 9-31).



LISTING 9-31: The execute() method for VerifyCredentialsCommand

```
public function execute(note:INotification):Void {
        var u:String = note.getBody()[0];
        var p:String = note.getBody()[1];

        var tProxy:TwitterProxy =
            TwitterProxy(facade.retrieveProxy(TwitterProxy.NAME));

        tProxy.initCall(TwitterProxy.VERIFY_CREDENTIALS, [u, p]);
}
```

www.it-ebooks.info

The `note.getBody()` value is set in `UserLoginMediator` and is sent with the `VERIFY_CREDENTIALS` notification, which you'll cover later.

Using Commands to Store and Retrieve Login Details

In Chapter 8 our `SharedObjectProxy` was specifically created to achieve saving and retrieving data from the last selected image when the Image Viewer app restarted. In both `RetrieveUserDetailsCommand` and `StoreLoginDetailsCommand`, `SharedObjectProxy` is also used. `SharedObjectProxy` is imported in both commands.

RetrieveUserDetailsCommand.as

With `RetrieveUserDetailsCommand` you are retrieving the username and password with which the user attempted to log in. In the `execute()` method you'll see that the command invokes the `init()` method for the `SharedObjectProxy` instance, supplying the "twitterClient" as a reference to the stored shared object (see Listing 9-32).



LISTING 9-32: The execute() method for RetrieveUserDetailsCommand

```
public function execute(note:INotification):Void {
        var sProxy:SharedObjectProxy =
            SharedObjectProxy(facade.retrieveProxy(SharedObjectProxy.NAME));
        sProxy.init("twitterClient");
}
```

StoreLoginDetailsCommand.as

While `RetrieveUserDetailsCommand` invokes the retrieval of the login data from `SharedObjectProxy`, `StoreLoginDetailsCommand` stores the login data to `SharedObjectProxy` via the `saveSO()` method. The login credentials are retrieved from `note.getBody()`, which is an array containing two values. The username is the first value, and the password is the second. The values are sent with the `AUTHENTICATE_USER` notification and are saved to `LoginProxy` (see Listing 9-33).



LISTING 9-33: The execute() method for StoreLoginDetailsCommand

```
public function execute(note:INotification):Void {
        var lProxy:LoginProxy = LoginProxy(facade.retrieveProxy(LoginProxy.NAME));
        lProxy.username = note.getBody()[0];
        lProxy.password = note.getBody()[1];

        var dataObj:Object = {username:lProxy.username, password: lProxy.password};

        var sProxy:SharedObjectProxy =
            SharedObjectProxy(facade.retrieveProxy(SharedObjectProxy.NAME));
        sProxy.init("twitterClient");
        sProxy.saveSO(dataObj);
}
```

www.it-ebooks.info

Using Commands to Retrieve Timelines, Retrieve Statuses, and Update User Profiles

Now take a look at each of the details contained in the `execute()` methods for the commands responsible for viewing timelines and statuses, starting with `ShowUserCommand`.

ShowUserCommand.as

`ShowUserCommand` will be triggered in response to a request to see the user's status. To use it, supply the static `String` variable `TwitterProxy.SHOW_USER` as the first argument for the `initCall()` method on the `TwitterProxy` instance. The second parameter is an array containing `tProxy.screenName` and two empty strings (see Listing 9-34).



LISTING 9-34: The execute() method for ShowUserCommand

```
public function execute(note:INotification):Void {
      var tProxy:TwitterProxy =
          TwitterProxy(facade.retrieveProxy(TwitterProxy.NAME));
      tProxy.initCall(TwitterProxy.SHOW_USER, [tProxy.screenName, "", ""]);

}
```

While this command is used for showing the authenticated user, you should realize that the command could also be used to show other Twitter users.

Next we'll take a look at `ShowFriendsTimelineCommand`.

ShowFriendsTimelineCommand.as

This command uses both `LoginProxy` and `TwitterProxy`. To receive friends' timelines for a user, our command `ShowFriendTimelineCommand` also needs to initialize the `initCall()` method on a `TwitterProxy` instance, this time passing `TwitterProxy.FRIENDS_TIMELINE` for the first argument and the number of parameters `params` for the second argument. The parameters include the screen name for the Twitter user as well as the login credentials (see Listing 9-35).



LISTING 9-35: The execute() method for ShowFriendsTimelineCommand

```
public function execute(note:INotification):Void {
      var tProxy:TwitterProxy = TwitterProxy(facade.retrieveProxy(TwitterProxy.NAME));
      var lProxy:LoginProxy = LoginProxy(facade.retrieveProxy(LoginProxy.NAME));
      var params:Array = [lProxy.username, lProxy.password, tProxy.screenName];
      tProxy.initCall(TwitterProxy.FRIENDS_TIMELINE, params);
}
```

Notice here that the last value in the `params` array is the Twitter `screenName`. While the majority of usernames can be used, it's not always guaranteed that the user will log in with his or her username. `TwitterProxy` is used for retrieving the current users, so here `screenName` will always be used.

www.it-ebooks.info

ShowUserTimelineCommand.as

While the user timeline isn't featured in the Twitter client, I've included `ShowUserTimelineCommand` here to show that there are really only two minor differences to the `execute()` method when compared with `ShowFriendsTimelineCommand`. The first difference is that the `params` value simply contains the username and password retrieved from the `LoginProxy`. There is no need for the `screenName` property. The second difference is that the static string variable `TwitterProxy.USER_TIMELINE` is supplied as the first argument for the `initCall()` method (see Listing 9-36).



LISTING 9-36: The execute() method for ShowUserTimelineCommand

```
public function execute(note:INotification):Void {
      var tProxy:TwitterProxy = TwitterProxy(facade.retrieveProxy(TwitterProxy.NAME));
      var lProxy:LoginProxy = LoginProxy(facade.retrieveProxy(LoginProxy.NAME));
      var params:Array = [lProxy.username, lProxy.password];

      tProxy.initCall(TwitterProxy.USER_TIMELINE, params);
}
```

UpdateStatusCommand.as

As with `ViewCredentialsCommand`, `UpdateStatusCommand` uses the value returned via `note.getBody()` as one of the arguments for the Twitter API call. This value is saved to the `status` variable, which is then passed onto the `params` array and should contain the text for the user's new Twitter post. The `username` and `password` retrieved from `LoginProxy` are also supplied as parameters. In `UpdateStatusCommand` the static string variable `TwitterProxy.UPDATE` is supplied as the first argument for `initCall()`, followed by the completed `params` (see Listing 9-37).



LISTING 9-37: The execute() method for UpdateStatusCommand

```
public function execute(note:INotification):Void {
      var tProxy:TwitterProxy = TwitterProxy(facade.retrieveProxy(TwitterProxy.NAME));
      var lProxy:LoginProxy = LoginProxy(facade.retrieveProxy(LoginProxy.NAME));
      var status:String = note.getBody().toString();
      var params:Array = [status, lProxy.username, lProxy.password];
      tProxy.initCall(TwitterProxy.UPDATE, params);
}
```

Having covered each of the commands used for our application, we can now move on to creating the view.

Exploring the View

In this section you'll examine the view defined for the application (see Figure 9-5).

www.it-ebooks.info

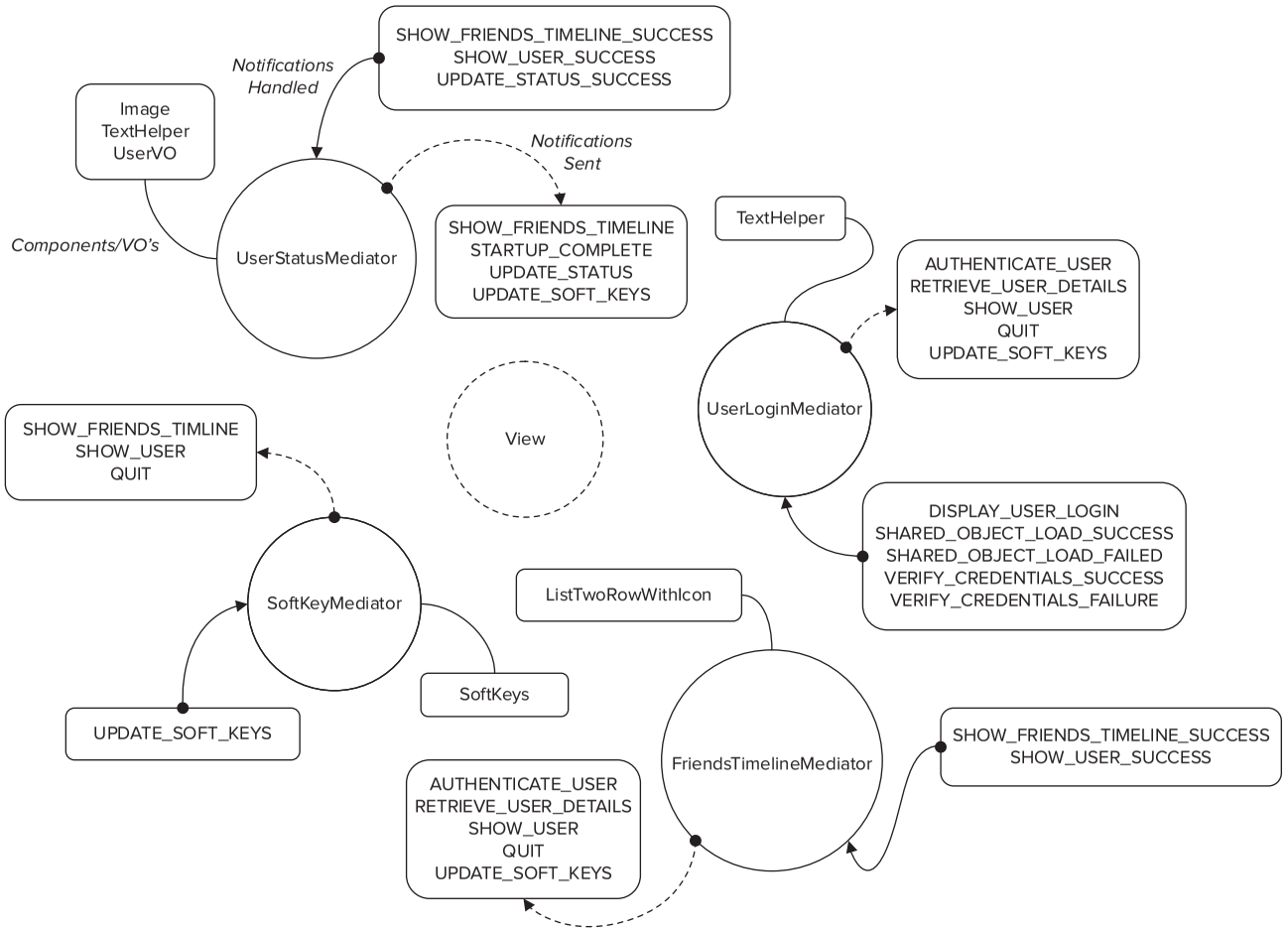


FIGURE 9-5

www.it-ebooks.info

In Figure 9-5 you can see three new mediator classes that are used for the Twitter client application:

- `UserLoginMediator`: for displaying user login
- `UserStatusMediator`: for displaying user status elements
- `FriendsTimelineMediator`: for displaying the timeline

The Twitter client will also use `SoftKeyMediator`, covered in earlier chapters.

The Project panel shown in Figure 9-6 indicates the file system layout for the view.



FIGURE 9-6

The User Login

In this section you examine `UserLoginMediator`, the mediator that allows our application to capture a user's Twitter credentials for authentication, allowing the user to use the main features of the application.

UserLoginMediator.as

`UserLoginMediator` is essentially a form consisting of a few `TextField` elements and a single Submit button.

Figure 9-7 shows our user login view for the Twitter client on the Nokia N95 8GB.

The mediator registers an interest in five notifications:

- `DISPLAY_USER_LOGIN`: for when a request is sent to view the user login
- `SHARED_OBJECT_LOAD_SUCCESS`: for when an attempt to retrieve the username and password stored on the device is successful
- `SHARED_OBJECT_LOAD_FAILED`: for when an attempt to retrieve the username and password stored on the device fails
- `VERIFY_CREDENTIALS_SUCCESS`: for when a login attempt has been deemed to be successful
- `VERIFY_CREDENTIALS_FAILURE`: for when a login attempt fails

Each notification is handled in the `handleNotification()` method. Before looking at how each notification is handled, let's look at the methods and components that make up this mediator.



FIGURE 9-7

www.it-ebooks.info

The core functions of the mediator are:

- `revealLogin()`: to show the login form
- `removeLogin()`: to remove the login form from view
- `updateSoftKeys()`: to update the soft keys
- `submitDetails()`: to submit login details
- `onSetFocusHandler()`: to detect when focus has moved to a text field
- `onKillFocusHandler()`: to detect when focus has moved from a text field

Let's first look at the `revealLogin()` method. The mediator consists of several `TextField` components for displaying input boxes and associated text labels, along with a Submit button. These are all rendered to the screen dynamically when the `revealLogin()` method is called. Five `TextField` instances are used: `statusLbl`, `usernameLbl`, `usernameTxt`, `passwordLbl`, and `passwordTxt`. The Submit button is called `submitBtn` (see Listing 9-38).



LISTING 9-38: The revealLogin() method for UserLoginMediator

```
public function revealLogin():Void {
     statusLbl = TextHelper.createLabel(txtFields, "sLbl", 11, 0, 250, 180, 30);
     usernameLbl = TextHelper.createLabel(txtFields, "uLbl", 16, 0, 67, 200, 30);
     usernameTxt = TextHelper.createInputField(txtFields, "uTxt", 18, 0, 90, 210, 30);
     usernameTxt.onKillFocus = Delegate.create(this, onSetFocusHandler);
     usernameTxt.onSetFocus = Delegate.create(this, onKillFocusHandler);
     passwordLbl = TextHelper.createLabel(txtFields, "pLbl", 16, 0, 127, 200, 30);
     usernameLbl.text = "Username or email address:";
     passwordLbl.text = "Password:";
     passwordTxt = TextHelper.createInputField   (txtFields, "pTxt", 18, 0, 150, 210, 30);
     passwordTxt.password = true;
     passwordTxt.onKillFocus = Delegate.create(this, onSetFocusHandler);
     passwordTxt.onSetFocus = Delegate.create(this, onKillFocusHandler);

     txtFields._x = Stage.width/2 – txtFields._width / 2;
     txtFields._y = 0;

     var submitBtnDepth:Number = _mc.getNextHighestDepth();
     submitBtn = _mc.attachMovie("PushButton", "sBtn", submitBtnDepth);
     submitBtn._x = 10;
     submitBtn._y = 200;
     submitBtn._labelText = "Sign in";
     submitBtn._textColor = 0xFFFFFF;
     submitBtn._backgroundColor = 0xFFFFFF;
     submitBtn._pressedBackgroundColor = 0x33CBFF;
     submitBtn._highlightColor = 0x33ABE9;
     submitBtn.onSelect = Delegate.create(this, submitDetails);
}
```

www.it-ebooks.info

The `statusLbl` is drawn and displays information relating to the user's login. This information is used to let the user know that the Twitter client is attempting a login, and subsequently displays an error message when the user fails to provide correct details. Should there be a problem with using the SWX PHP service or if there is a timeout on the service, the appropriate message will be displayed in `statusLbl`.

To add `statusLbl` to the view, use the following code:




```
statusLbl = TextHelper.createLabel(txtFields, "sLbl", 11, 0, 250, 180, 30);
```

code snippet TextHelper.as

You'll notice in this line of code that `userLoginMediator` uses the `TextHelper` class to create `statusLbl`.

In the `TextHelper` class are two public static methods, `createLabel()` and `createInputField()`. For creating static text labels (that don't require user input), use the `TextHelper.createLabel()` method, and for text fields that do require user input, use the `TextHelper.createInputField()` method. Because the application needs to display numerous `TextField` components, it makes sense to have the `TextHelper` class, simply to aid development. All the styles have been defined for the Twitter client, so if you reuse `TextHelper`, be sure to check out the styles.

Let's now return to our `statusLbl`. Seven parameters need to be supplied to the `TextHelper.createLabel()` method: a reference `MovieClip` container, a unique instance name, the font size, and the `_x`, `_y`, `_width`, and `_height` properties.

`statusLbl` will be created in the `txtFields` container along with the other text fields in the view. Specifically, `statusLbl` will have an instance name of "sLbl", have a font size of 11, be aligned to 0 along the *x* axis, be aligned to 250 along the *y* axis, and have a width of 180 and a height of 30 pixels.

`usernameLbl` and `passwordLbl` are both created in the same way but with different parameters:




```
usernameLbl = TextHelper.createLabel(txtFields, "uLbl", 16, 0, 67, 200, 30);
passwordLbl = TextHelper.createLabel(txtFields, "pLbl", 16, 0, 127, 200, 30);
```

code snippet UserLoginMediator.as

Both these fields have their font sizes set to 16. This differs from the `statusLbl` text size, and because these fields don't need to be dynamic, the `text` values are set:




```
usernameLbl.text = "Username or email address:";
passwordLbl.text = "Password:";
```

code snippet UserLoginMediator.as

www.it-ebooks.info

The remaining `TextField` components created via `revealLogin()` are `usernameTxt` and `passwordTxt`. Both require user input, so they are created using the `TextHelper.createInputField()` method:




```
usernameTxt = TextHelper.createInputField(txtFields, "uTxt", 18, 0, 90, 210, 30);
passwordTxt = TextHelper.createInputField(txtFields, "pTxt", 18, 0, 150, 210, 30);
```

code snippet UserLoginMediator.as

The `createInputField()` method requires the same parameters as `createLabel()`. Notice that the `password` property on `passwordTxt` is also set to `true`; this ensures that the user's password is hidden by asterisks:




```
passwordTxt.password = true;
```

code snippet UserLoginMediator.as

On both `usernameTxt` and `passwordTxt` the `onSetFocus` and `onKillFocus` events are handled, by `onSetFocusHandler()` and `onKillFocusHandler()` respectively:




```
usernameTxt.onKillFocus = Delegate.create(this, onSetFocusHandler);
usernameTxt.onSetFocus = Delegate.create(this, onKillFocusHandler);
passwordTxt.onKillFocus = Delegate.create(this, onSetFocusHandler);
passwordTxt.onSetFocus = Delegate.create(this, onKillFocusHandler);
```

code snippet UserLoginMediator.as

The aim of defining these methods is to detect when the user highlights an input text field. The `onSetFocus` event handler method is invoked when the user highlights either `usernameTxt` or `passwordTxt`, whereas the `onKillFocus` event handler method is invoked when the user moves from the text field. Both handlers change the `backgroundColor` property of the text field; the handler returns the instance of the text field that has invoked the method (see Listing 9-39).



LISTING 9-39: The onSetFocus and onKillFocus event handlers for UserLoginMediator

```
public function onSetFocusHandler(textField:TextField):Void {
      textField.backgroundColor = 0xFAFFBD;
}

public function onKillFocusHandler(textField:TextField):Void {
      textField.backgroundColor = 0xFFFFFF;
}
```

www.it-ebooks.info

Once each of the text components has been rendered to the `MovieClip` reference container `txtFields`, the container is aligned to the stage:




```
txtFields._x = Stage.width/2 – txtFields._width / 2;
txtFields._y = 0;
```

code snippet UserLoginMediator.as

One part of the "reveal" for the mediator is creating the `sumitBtn`, which is an instance of the `PushButton` component covered in Chapter 4. Note that when the user clicks the Submit button, the `submitDetails()` method is called:




```
submitBtn.onSelect = Delegate.create(this, submitDetails);
```

code snippet UserLoginMediator.as

The `submitDetails()` method simply sends the `text` from `usernameTxt.text` and `passwordTxt.text`. When `submitDetails()` is called, the push button `submitBtn` is disabled and our status `statusLbl.text` is set to display "Attempting to verify user." The notification `AUTHENTICATE_USER` is then sent via `sendNotification()` with the notification body containing a reference to the `userCredentials` array (see Listing 9-40).



LISTING 9-40: The submitDetails() method for UserLoginMediator

```
private function submitDetails():Void {
      var userCredentials:Array = [usernameTxt.text, passwordTxt.text];
      statusLbl.text = "Attempting to verify user.";
      submitBtn._isEnabled = false;
      sendNotification(ApplicationFacade.AUTHENTICATE_USER, userCredentials)
}
```

You now should have covered the `revealLogin()`, `submitDetails()`, `onSetFocusHandler()`, and `onKillFocusHandler()` methods. There are two more methods to cover in `UserLoginMediator` before we examine our notification handler: `removeLogin()` and `updateSoftKeys()`.

For the `removeLogin()` method, the reference container for each `TextField`, `txtFields`, and `submitBtn` is removed from view, using a combination of `removeMovieClip()` and `removeTextField()`.

In the `updateSoftKeys()` method, the left and right soft key labels are defined. The left soft key label is set "Quit" and is mapped to the `QUIT`, allowing the user to exit the application (see Listing 9-41).



LISTING 9-41: The updateSoftKeys() method for UserLoginMediator

```
public function updateSoftKeys():Void {
      var msk:Object = {label:"", note:""};
```

www.it-ebooks.info

```
        var lsk:Object = {label:" Quit", note:ApplicationFacade.QUIT};
        var rsk:Object = {label:"", note:""};
        sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
    }
```

Finally, take a look at the `handleNotification()` method (see Listing 9-42).



LISTING 9-42: The handleNotification() method for UserLoginMediator

```
public function handleNotification(note:INotification):Void {
        switch (note.getName()) {
                case ApplicationFacade.DISPLAY_USER_LOGIN:
                        updateSoftKeys();
                        revealLogin();
                        sendNotification(ApplicationFacade.RETRIEVE_USER_DETAILS);
                break;
                case ApplicationFacade.VERIFY_CREDENTIALS_SUCCESS:
                        removeLogin();
                        sendNotification(ApplicationFacade.SHOW_USER);
                break;
                case ApplicationFacade.VERIFY_CREDENTIALS_FAILURE:
                        var textFormat = new TextFormat();
                        textFormat.color = 0xFF0000;
                        statusLbl.setNewTextFormat(textFormat);
                        statusLbl.text = note.getBody().toString();
                        submitBtn._isEnabled = true;
                break;
                case ApplicationFacade.SHARED_OBJECT_LOAD_SUCCESS:
                        usernameTxt.text = note.getBody().username;
                        passwordTxt.text = note.getBody().password;
                        submitBtn._isEnabled = true;
                break;
                case ApplicationFacade.SHARED_OBJECT_LOAD_FAILED:
                        statusLbl.text = "Enter Username and Password.";
                        submitBtn._isEnabled = true;
                break;
        }
}
```

You'll see in the switch statement that when the notification `DISPLAY_USER_LOGIN` is sent, the methods `updateSoftKeys()` and `revealLogin()` are called, and the mediator then sends the `RETRIEVE_USER_DETAILS` notification via `sendNotification`.

When the `VERIFY_CREDENTIALS_SUCCESS` notification is received, the `removeLogin()` method is called. The mediator then sends the `SHOW_USER` notification. Remember that `VERIFY_CREDENTIALS_SUCCESS` is sent by `TwitterProxy` only when SWX PHP successfully returns user details after the call to `verifyCredentials` is made. Alternatively, when the authentication fails, the `VERIFY_CREDENTIALS_FAILURE` notification is received. The mediator handles this by setting the `statusLbl` color to red (`0xFF0000`) after creating an instance of `TextFormat` and applying it through `setNewTextFormat()`. It then applies the text received in `note.getBody()` to `statusLbl.text`.

www.it-ebooks.info

The mediator then re-enables the `submitBtn`, allowing the user to resubmit. You'll recall that the button is disabled when the user submits his or her details.

The `SHARED_OBJECT_LOAD_SUCCESS` notification is also handled by `UserLoginMediator`. Remember that `RETRIEVE_USER_DETAILS` is sent by the mediator when it receives the `DISPLAY_USER_LOGIN` notification. The `RETRIEVE_USER_DETAILS` notification is mapped to the `RetrieveUserDetailsCommand` and subsequently initializes `SharedObjectProxy` for the Twitter client. When the proxy has data, it sends it via the notification body to the mediator, which assigns the username and password properties to `usernameTxt.text` and `passwordTxt.text`:

Available for download on Wrox.com

```
usernameTxt.text = note.getBody().username;
passwordTxt.text = note.getBody().password;
```

code snippet UserLoginMediator.as

Finally, when the shared object fails to load any data, the `SHARED_OBJECT_LOAD_FAILED` notification is sent and the mediator sets the `statusLbl.text` to the text "Enter Username and Password."

That completes the `UserLoginMediator` class.

Displaying a User's Twitter Status

The user's Twitter status view follows the login. For displaying the user's status details, the mediator `UserStatusMediator` is used.

Figure 9-8 shows our user's Twitter status view for the Twitter client on the Nokia N95 8GB.

This mediator will contain the elements to display a profile image and show the user's latest tweet and the number of friends and followers the user has. From this section of the application the user will also be able to update his or her status and navigate to friends' timeline.



FIGURE 9-8

UserStatusMediator.as

`UserStatusMediator` contains a few elements: an image called `profileImg`, a push button called `updateBtn`, and several text fields that are contained in a reference `MovieClip` container called `txtFields`.

The `onRegister()` method instantiates a new `Image` and aligns it 12 pixels along the *x* and *y* axis (see Listing 9-43).

LISTING 9-43: The onRegister() method for UserStatusMediator

Available for download on Wrox.com

```
public function onRegister():Void {
      profileImg = new Image(_mc, 48, 48);
      profileImg.moveTo(12, 12);
```

www.it-ebooks.info

```
        txtFields = _mc.createEmptyMovieClip("textFields", _mc.getNextHighestDepth());
    }
```

You covered creating input text and labels in the last section. That portion of the code review is not repeated here; however, several `TextField` components are used in `UserStatusMediator`, including seven static label fields: `screenNameLbl`, `followersLbl`, `followingLbl`, `statusesLbl`, `locationLbl`, `latestLbl`, and `updateLbl`, and one input text field, `latestTxt`.

The user's screen name should be displayed in `screenNameLbl`, while `locationLbl` should display the user's location. `followersLbl` should show the total number of followers the user has, while `followingLbl` should display the total number of users the user is following. The total number of statuses a user has made is displayed in `statusesLbl`. `updateLbl` should show the status of the update when the user has attempted to update his or her profile, and, finally, `latestTxt` should display the user's latest Twitter update.

At this point you should realize that the text fields will contain specific details returned from `TwitterProxy`, so let's look at how these fields are populated.

`UserStatusMediator` has three notification interests:

- `SHOW_USER_SUCCESS`: for displaying the user's status
- `SHOW_FRIENDS_TIMELINE_SUCCESS`: for removing the mediator when a request for the friends timeline has been successfully made
- `UPDATE_STATUS_SUCCESS`: for displaying updates to the user's status in the mediator

Each notification is handled in the `handleNotification()` method (see Listing 9-44).



LISTING 9-44: The handleNotification() method for UserStatusMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_USER_SUCCESS:
                  updateSoftKeys();
                  revealProfile(UserVO(note.getBody()));
            break;
            case ApplicationFacade.UPDATE_STATUS_SUCCESS:
                  updateBtn._isEnabled = true;
                  updateLbl.text = note.getBody().toString();
            case ApplicationFacade.SHOW_FRIENDS_TIMELINE_SUCCESS:
                  removeProfile();
            break;
      }
}
```

The `updateSoftKeys()` method is called when the mediator receives the `SHOW_USER_SUCCESS` notification. Here the left and right soft key labels are defined. The left soft key label is set to "Friends" and is mapped to the `SHOW_FRIENDS_TIMELINE` notification, allowing the end user to view his or her friends' timeline in the application. The right soft key label is set to "Back" and is mapped

www.it-ebooks.info

to the STARTUP_COMPLETE notification, allowing the end user to navigate back to his or her login view (see Listing 9-45).



LISTING 9-45: The updateSoftKeys() method for UserStatusMediator

```
public function updateSoftKeys():Void {
      var msk:Object = {label:"", note:""};
      var lsk:Object = {label:" Friends", note:ApplicationFacade.SHOW_FRIENDS_
          TIMELINE};
      var rsk:Object = {label:" Back", note:ApplicationFacade.STARTUP_COMPLETE};

      sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
}
```

When the notification SHOW_USER_SUCCESS is sent from TwitterProxy, the proxy sends an instance of UserVO, a value object in the notification body that contains the user's details. The revealProfile() method takes the UserVO received from note.getBody() as the parameter and then assigns values from the value object to the mediator's components.

In examining revealProfile(), you will see that this is where each TextField component is created. Here each of the properties from the UserVO instance user is assigned to its corresponding component:




```
latestLbl.text = "What are you doing?";
latestTxt.text = user.status;
screenNameLbl.text = user.screenName;
locationLbl.text = user.location;
followingLbl.text = user.friends + newline + "following";
followersLbl.text = user.followers + newline + "followers";
statusesLbl.text = user.updates + newline + "tweets";
```

code snippet UserStatusMediator.as

Following the text assignments, the image path imageUrl is assigned to profileImg.dataProvider, initializing the image load, as covered in Chapter 8. The frame surrounding the image is also drawn:




```
profileImg.dataProvider = user.profile.imageUrl;
profileImg.drawFrame(2, 0x7EBAC0);
```

code snippet UserStatusMediator.as

The final portion of revealProfile() displays the push button, called updateBtn. When the user presses the Update button, the updateStatus() method is invoked. In updateStatus(), the updateLbl.text property is set to display the message "Updating please wait," while the Update button itself is disabled, preventing the user from calling the method again.

www.it-ebooks.info

Figure 9-9 shows the screen that is displayed when a user updates his or her profile.



FIGURE 9-9

The `updateStatus()` method also sends the notification `UPDATE_STATUS`, which sends the text contained in the `latestTxt.text` property (see Listing 9-46).

Available for download on Wrox.com

LISTING 9-46: The updateStatus() method for UserStatusMediator

```
private function updateStatus():Void {
      updateLbl.text = "Updating please wait.";
      updateBtn._isEnabled = false;
      sendNotification(ApplicationFacade.UPDATE_STATUS, latestTxt.text);
}
```

When the notification `UPDATE_STATUS_SUCCESS` is handled by the mediator, the Update button is enabled again:

Available for download on Wrox.com

```
updateBtn._isEnabled = true;
updateLbl.text = note.getBody().toString();
```

code snippet UserStatusMediator.as

www.it-ebooks.info

Finally, the mediator is also interested in `FRIENDS_TIMELINE_SUCCESS`, which is received by the mediator when a successful call to view the friends timeline is received from `TwitterProxy`. When `FRIENDS_TIMELINE_SUCCESS` is retrieved, the `removeProfile()` method is called. This method removes each `TextField` component from view and also removes the Update status button and the profile image (see Listing 9-47).



LISTING 9-47: The revealProfile method for UserStatusMediator

```
public function removeProfile():Void {
      screenNameLbl.removeTextField();
      updateLbl.removeTextField();
      locationLbl.removeTextField();
      followingLbl.removeTextField();
      followersLbl.removeTextField();
      statusesLbl.removeTextField();
      latestLbl.removeTextField();
      latestTxt.removeTextField();
      updateBtn.removeMovieClip();
      profileImg.destroy();
}
```

You've now covered each aspect of `UserStatusMediator`. In the process you've learned how to display a Twitter user's details and how to submit and handle Twitter status updates. Next we'll take a look at viewing the friends timeline.

Viewing the Friends Timeline

`FriendsTimelineMediator` displays the end user's friends timeline, using the `ListTwoRowWithIcon` component.

In Figure 9-10 you can see the friends timeline view.



FIGURE 9-10

FriendsTimelineMediator.as

`FriendsTimelineMediator` is interested in two notifications:

- `SHOW_FRIENDS_TIMELINE_SUCCESS`: for when `TwitterProxy` successfully retrieves details from the SWX Twitter API call `friendsTimeline`
- `SHOW_USER_SUCCESS`: for when the `TwitterProxy` receives details from the SWX Twitter API call `showUser`

This mediator consists of a single view list called `friendsTimeline`, a `MovieClip` that will attach a `ListTwoRowWithIcon` component from the library. `FriendsTimelineMediator` has two core methods: `revealTimeline()` and `updateSoftKeys()`. The `onRegister()` method creates `friendsTimeline` when the mediator is registered.

www.it-ebooks.info

The `revealTimeline()` method simply attaches the list component if it isn't already present and then styles it (see Listing 9-48).



LISTING 9-48: The revealTimeline() method for FriendsTimelineMediator

```
public function revealTimeline():Void {
      if(!friendsTimeline){
            var depth:Number = timeline.getNextHighestDepth();
            friendsTimeline = timeline.attachMovie("ListTwoRowWithIcon", "timeline",
                              depth);
            friendsTimeline._y = 0;
            friendsTimeline._height = 320;
            friendsTimeline._textColor = 0xFFFFFF;
            friendsTimeline._backgroundColor = 0xFFFFFF;
            friendsTimeline._scrollbarVisibility = false;
            friendsTimeline._pressedBackgroundColor = 0x33CBFF;
            friendsTimeline._highlightColor = 0x33CBFF;
            friendsTimeline._scrollSliderColor = 0x33CBFF;
      }
}
```

The `updateSoftKeys()` method sets the labels for the left and right soft keys and assigns their respective notification names. `SoftKeyMediator` is then updated. The "Refresh" label assigned to the left soft key sets the `SHOW_FRIENDS_TIMELINE` notification to the soft keys, while the "Back" label assigned to the right soft key sets the `SHOW_USER` notification (see Listing 9-49).



LISTING 9-49: The updateSoftKeys() method for FriendsTimelineMediator

```
public function updateSoftKeys():Void {
      var msk:Object = {label:"", note:""};
      var lsk:Object = {label:"Refresh", note:ApplicationFacade.SHOW_FRIENDS_
                        TIMELINE};
      var rsk:Object = {label:"Back", note:ApplicationFacade.SHOW_USER};

      sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
}
```

The `SHOW_FRIENDS_TIMELINE_SUCCESS` notification is handled in the `handleNotification()` method where the call to `updateSoftKeys()` is made, followed by the call to the `revealTimeline()` method. The `note.getBody()` array then assigns the first and second lines of the list component. Remember that `TwitterProxy` collates the `screenNames` and `statusUpdates`. Here `note.getBody()[1]`, which contains our `screenNames`, is assigned to `timeline._text1` value in the `ListTwoRowWithIcon` component, while `note.getBody()[2]`, which contains our `statusUpdates`, is set to `_text2` (see Listing 9-50).

www.it-ebooks.info



LISTING 9-50: The handleNotification() method for FriendsTimelineMediator

```
public function handleNotification(note:INotification):Void {
      switch(note.getName()) {
            case ApplicationFacade.SHOW_FRIENDS_TIMELINE_SUCCESS:
                  updateSoftKeys();
                  revealTimeline();
                  friendsTimeline._text1 = note.getBody()[1];
                  friendsTimeline._text2 = note.getBody()[2];
            break;
            case ApplicationFacade.SHOW_USER_SUCCESS:
                  friendsTimeline.removeMovieClip();
                  delete friendsTimeline;
            break;
      }
}
```



Initially `note.getBody()[0]`*, which contained the profile images, was assigned to the* `_icon` *property of the* `ListTwoRowWithIcon` *component. However, the images for each profile in the list became rather distorted, so at the final edit this was removed.*

In `handleNotification()` you'll also see that `SHOW_USER_SUCCESS` notification is handled to remove the list component from view.

Viewing the Application in Device Central

You have now covered each aspect of the Twitter client. If you haven't done so already, run the application in Device Central. Ensure that you have selected the Nokia N95 8GB for testing.

SUMMARY

In this chapter you explored the Twitter API and learned about the SWX format and the code behind the Twitter client.

You learned how to use various elements from the very popular Twitter API. You also learned about Aral Balkan's SWX PHP service, which provides a simple way of using data from the Twitter API service in Flash-enabled mobile applications.

Using the SWX service isn't limited to the SWX Twitter API, and I encourage you to have a go at building your own custom services to perhaps extend some of the apps you've covered so far in the book, of course incorporating SWX.

In the next chapter you'll take a look at a different set of API's that allow you to extend mobile applications outside the realm of mostly Nokia Flash Lite *enabled* devices.

www.it-ebooks.info

10

Using Nokia's S60 Platform Services API

WHAT'S IN THIS CHAPTER?

- Listing installed applications
- Importing calendar events
- Managing contacts
- Retrieving a device's coordinates
- Sending text messages

In this chapter, you will learn about how you can extend the features of your Flash Lite mobile applications using the libraries from Forum Nokia and the S60 Platform Services API. You will be provided with snippets and code examples that you can use in the development of your applications, while comprehensively exploring the APIs.

The S60 Platform Services API library, which is referenced here, can be found on the Forum Nokia Developers website at `http://library.forum.nokia.com/`.

S60 PLATFORM SERVICES OVERVIEW

The S60 Platform is the software platform specially created for mobile phones that runs on Symbian OS, one of the leading smartphone platforms in the world. You'll frequently come across references to the platform in various guises. For one, there have been several releases of the platform over the years, including: S60 (2001), S60 2nd Edition (2003), S60 3rd Edition (2005), and S60 5th Edition (2008). Between each major release and the next, there have

www.it-ebooks.info

been iterations containing significant changes or features that were updated in the platform, such as support for higher screen resolutions and increased security for installing third-party applications. One of the features introduced in S60 5th Edition was support for a higher screen resolution of 640 x 360 px, with touch input support.

The S60 5th Edition also introduced specific ActionScript extensions for Flash Lite 3.0 that give you access to specific phone features such as calendars, contacts, messaging, and sensors.

In this section, you'll review some of the common elements in the S60 Platform Services that are found in the *How to Use the API Methods* section.



At the time of writing, the S60 Platform Services were supported on the devices running S60 5th Edition Symbian OS, which runs Flash Lite 3.0. For each of the examples, it is envisaged that you will have a supported device to run content on.

Download and Install the Library

Before you start to use the services, you need to download and install the ActionScript S60 Platform Services API Library. The `.zip` file can be downloaded from the Forum Nokia website at `http://library.forum.nokia.com/topic/Flash_Lite_Developers_Library/ActionScript_library_for_S60_Platform_Services.zip`

Depending on your platform, you also need to download the appropriate folder:

- **For Mac** — Copy the extracted folder to `/Users/<user name>/Library/Application/Support/Adobe/Flash CS4/en/Configuration/Classes/`.
- **For Windows Vista** — Extract the folder to `/Users/<user name>/AppData/Local/Adobe/Flash CS4/en/Configuration/Classes`.
- **For Windows XP** — Extract the folder to `/<user name>/Local Settings/Application Data/Adobe/Flash CS4/en/Configuration/Classes/`.

You should now be ready to progress through the examples in this chapter.

How to Use the API Methods

Each service's API call requires two parameters: the name of the service provider and an interface. In Table 10-1 you'll see a current list of S60 Platform Services APIs available. In the table you'll see the list of Service API names with their corresponding provider, which is its class reference, and the name of the interface required; both are used for instantiating their respective service.

www.it-ebooks.info

TABLE 10-1: S60 Platform Services

SERVICE API	PROVIDER	INTERFACE
AppManager	Service.AppManager	IAppManager
Calendar	Service.Calendar	IDataSource
Contacts	Service.Contact	IDataSource
Landmarks	Service.Landmarks	IDataSource
Location	Service.Location	ILocation
Logging	Service.Logging	IDataSource
Media Management	Service.MediaManagement	IDataSource
Messaging	Service.Messaging	IMessaging
Sensor	Service.Sensor	ISensor
SystemInfo	Service.SysInfo	ISysInfo

Source: Forum Nokia ActionScript Service object page at `http://library.forum.nokia.com/index.jsp?topic=/Flash_Lite_Developers_Library/GUID-BB53CD32-F898-418C-BA1D-013E7678C8C9.html`

There are two types of methods within the API as specified by Forum Nokia Library: *synchronous* and *asynchronous*. Understanding these forms a small but significant part of learning the libraries and syntax of each of the methods.

Synchronous Methods

The method signature for assigning synchronous methods of the Services API requires you to supply a single input parameter object as an argument. For the examples given in this chapter, you'll notice that the APIs specify the parameter object as `params` and have specific properties. When you call a synchronous method, it returns the result of the method instantly (see Listing 10-1).



LISTING 10-1: Calling a synchronous method

```
import com.nokia.lib.Service;

var messaging = new Service("Service.Messaging", "IMessaging");

var params:Object = { MessageType:"SMS",
                      To:"recipient number 1.",
                      BodyText:"Hello World" };

var result:Object = messaging.Send(params);
```

www.it-ebooks.info

A service call returns an object; in Listing 10-1 and throughout this chapter, this is assigned to `result`. This can have up to three properties defined:

- `ReturnValue` — This is an object specific to the Services API method being called. Some methods do not return this property, but the majority will. You'll learn more about this object for each method throughout this chapter.
- `ErrorCode` — This is a number defining a specific error that occurs from a method call. These are not discussed at length, but a few examples are provided for reference.
- `ErrorMessage` — This is a string representation of an error and `ErrorCode`. This property is only defined when an error has occurred.

Asynchronous Methods

For asynchronous methods you need to supply a second argument in addition to `params`. The `callback` argument is essentially a reference to an event handler method. The handler is invoked when the result or requested information is retrieved from the service call.

There are three properties that you can obtain from within each callback method:

- `transactionID` — A unique number assigned to the `callback` invoked.
- `eventID` — A string representation for the status code of the current service `callback` and `transactionID`. This can be any one of the following seven string status code values, relating to the asynchronous service call request:
  - `completed` — The service call is completed.
  - `cancel` — The service call is canceled.
  - `error` — An error occurs during the service call.
  - `inprogress` — The service call is currently in progress.
  - `started` — A service call has just started.
  - `stopped` — A service call has just stopped.
  - `unknown` — A service call invoked an unknown event.
- `result` — An object, which can consist of the three properties returned for synchronous calls in the Services API: `Returnvalue`, `ErrorCode`, and `ErrorMessage`.

Listing 10-2 demonstrates a way in which you can define an asynchronous method with a callback handler.



LISTING 10-2: Calling an asynchronous method

```
import com.nokia.lib.Service;

var messaging = new Service("Service.Messaging", "IMessaging");
```

www.it-ebooks.info

```
var params:Object = { MessageType:"SMS",
                      To:"recipient number 1.",
                      BodyText:"Hello World" };

var result:Object = messaging.Send(params, onSend);

function onSend(transactionID:Number, eventID:String, result:Object){

}
```

Canceling Asynchronous Method Calls

With asynchronous calls there may be a small time delay in retrieving results, and you may want to simply cancel the request. Each API has a `Cancel()` method specifically for this purpose, an example of which is shown in Listing 10-3. Here, the `TransactionID` is retrieved from the `result` object, which can be obtained from asynchronous method calls across the API. The `transactionID` is then passed to the `Cancel()` method.



LISTING 10-3: Canceling an asynchronous call

```
import com.nokia.lib.Service;

var messaging = new Service("Service.Messaging", "IMessaging");

var params:Object = { MessageType:"SMS",
                      To:"recipient number 1.",
                      BodyText:"Hello World" };

var result:Object = messaging.Send(params, onSend);

var transactionID:Number = result.TransactionID;
var hasCancelled:Boolean = messaging.Cancel(transactionID);

function onSend(transactionID:Number, eventID:String, result:Object){

}
```



There are some methods in the S60 Services API that give you the option of calling either the synchronous or asynchronous implementation. When implementing either of the services, you should consider the type of call you want to make as this will impact the overall structure of your code. For the majority of the examples in this chapter, the synchronous implementation is used.

The ReturnValue

The `ReturnValue` is where you will obtain the results of your synchronous or asynchronous method calls. Each method in the API will provide a different `ReturnValue` depending on the service call implemented.

www.it-ebooks.info

Defining Filter and Sort Objects

The majority of the methods in the API allow you to set the criteria for the results returned in `ReturnValue`. By using a `Filter` object property in the parameters for the call, you can specify what results should be returned. Using the `Sort` object property you can also *order* the results returned by the call.

Both these objects are handy, especially where the data you require is complex.

Error Codes

Every service API call returns an `ErrorCode` and `ErrorMessage` property, whether synchronous or asynchronous. These are not covered by this chapter in depth; however, in Table 10-2 you'll see a brief list of some of the service API error codes found in the S60 Platform Services for Flash Lite, with their corresponding description.

TABLE 10-2: Service API error codes

ERROR CODE	DESCRIPTION
-304	General error This error occurs if the transaction ID returned by an asynchronous call is -1.
0	Success
1000	Invalid service argument
1001	Unknown argument name

Listing 10-4 demonstrates how to retrieve `ErrorCode` and `ErrorMessage` from a synchronous call to the Messaging Service API.



LISTING 10-4: Retrieving ErrorCode and ErrorMessage properties from a synchronous call

```
import com.nokia.lib.Service;

var messaging = new Service("Service.Messaging", "IMessaging");

var params:Object = { MessageType:"SMS",
                      To:"recipient number 1.",
                      BodyText:"Hello World" };

var result:Object = messaging.Send(params);
var errorCode:Number = result.ErrorCode;
var errorMsg:String = result.ErrorMessage;
```

www.it-ebooks.info

Having covered the common aspects of the APIs, you'll now explore using the S60 service APIs, first taking a look at the AppManager Service API.

USING APPMANAGER

With the AppManager Service API, you can launch applications or documents that are external to the Flash Lite player, while your application is still running. You can also use the API to open preexisting documents or create new ones.

API Features

There are four core methods that you can use in the AppManager Service API:

- `GetList()` — To retrieve a list of applications
- `LaunchApp()` — To launch an application
- `LaunchDoc()` — To launch a document
- `Cancel()` — To cancel an asynchronous method call

In the following subsections, you cover several functional aspects of using AppManager, including:

- Retrieving information about installed apps
- Distinguishing between types of installed apps
- Filtering information on installed apps
- Handling results from the `GetList()` call
- Launching applications and documents

Retrieving Information on Installed Apps

Use the `GetList()` method when you want to retrieve information about applications on a mobile device, for pre-installed or user-installed applications. To specify which types of applications you want to retrieve information about, you need to supply a single object parameter, which can have two properties defined: `Type` and `Filter`. Together these define the information returned about applications on the mobile device.

In Listing 10-5 you can see that the `Service` class is imported and instantiated using `Service.AppManager` for the service provider name and then `IAppManager` for the interface reference. Once you instantiate the `Service` object, you can use the `GetList()` method to retrieve specific information about the applications installed on the device, including the application's name, version, and file path. You can then use these properties to launch documents, services, or the applications from within the Flash Lite application. Here, the `Type` property is set to `Application`.

www.it-ebooks.info


Available for download on Wrox.com

LISTING 10-5: Using the GetList method to retrieve a list of applications

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var params:Object = { Type:"Application" };
var result:Object = appManager.GetList(params);
var appList:Array = result.ReturnValue;
```

Distinguishing between Types of Apps

Defining the `Type` property allows you to retrieve *all* applications installed on the device via the `Application` string, as previously mentioned, or "user-installed" applications via the `UserInstalledPackage` string value (see Listing 10-6).


Available for download on Wrox.com

LISTING 10-6: Using the GetList method to retrieve information on user installed packages

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var params:Object = { Type:"UserInstalledPackage" };
var result:Object = appManager.GetList(params);
var appList:Array = result.ReturnValue;
```

Filtering Information on Installed Apps

The `Filter` property can be set to define a criterion for selecting a more specific range of applications that you want to receive information about, and can only be used when you have specified the `Type` property as `Application`. The object has two properties: `MIMEType` and `DocumentPath`. Both properties are strings, where the `Filter.DocumentPath` property references the path of the document you want to base your application filter on. The `Filter.MIMEType` is the MIME type that an application understands. The `Filter` object should only have either the `MIMEType` or `DocumentPath` defined. If both the `DocumentPath` and the `MIMEType` are specified, the filter will be based on the `DocumentPath`.


Available for download on Wrox.com

LISTING 10-7: Using a Filter object to retieve GetList method to return information on all applications

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var filter:Object = { MimeType:"image/jpeg" };
var params:Object = { Type:"Application", Filter:filter };
var result:Object = appManager.GetList(params);
var appList:Array = result.ReturnValue;
```

Handling Results from the GetList() Call

If you specify the `Type` property as `Application`, the `ReturnValue` property will return four parameters:

www.it-ebooks.info

- `Uid` — A string that contains a unique ID for the application binary (EXE or DLL)
- `Path` — A string that contains the path of the application; for example, the Calculator application is found at `c:\sys\bin\calculator.exe`.
- `Caption` — A string that contains the title of the application
- `ShortCaption` — A string that contains the short title of the application. For example, the short caption may be displayed beneath an icon on the mobile device.

If you specify the `Type` property as `UserInstalledPackage`, the `ReturnValue` property will have five parameters:

- `PackageName` — A string that contains the name of the application
- `Version` — A string that contains the version of the application
- `Uid` — A string that contains a unique ID for the application binary (EXE or DLL)
- `Vendor` — A string that contains the vendor of the application
- `Drive` — A string that contains the drive where the application is installed

In Listing 10-8 you'll see that the `Caption`, `Uid`, and `Path` properties are retrieved from the AppManager's `GetList()` call and set to the variable `info`.


Available for download on Wrox.com

LISTING 10-8: Retrieving application properties using the GetList method

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var params:Object = { Type:"Application" };
var result:Object = appManager.GetList(params);
var appList:Array = result.ReturnValue;
var appEntry:Object;
var info:String;

do {

      appEntry = appList.next();

      if (null != appEntry) {

            var caption:String = appEntry.Caption;
            var uid:String = appEntry.Uid;
            var path:String = appEntry.Path;

            info += caption + " | " +  uid + " | " + path + newline;

      } else {

            info += "Total Applications = " + appList.length;

            break;

      }

} while (true);
```

www.it-ebooks.info

To take real advantage of this API, you need to know a little about the applications and services installed on your Nokia Device; this will allow you to interact with documents and launch applications. Table 10-3 shows some examples of application captions and their corresponding UID values that were retrieved by running the `GetList()` method on the Nokia N97.

TABLE 10-3: Pre-installed applications for the Nokia N97

APPLICATION	UID
Location	0x10283139
Landmarks	0x101f85a2
Contacts	0x200a94c
Maps	0x20001f63
App. manager	0x101f8512
File manager	0x101f84eb
Calendar	0x10005901
Calculator	0x10005902

Take note of the UID values for the applications listed in Table 10-3 as these are key to launching applications. The UID needs to be supplied to the `LaunchApp()` and `LaunchDoc()` methods of the API.

Launching Applications and Documents

Listing 10-9 demonstrates launching the Calculator application using the UID listed in Table 10-3. The parameter object for the call, `params`, has an `ApplicationId` property defined as `s60uid://0x10005902`. You should notice that this contains the UID for the Calculator app.



LISTING 10-9: Launching the Calculator application with LaunchApp

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var params:Object = { ApplicationId:"s60uid://0x10005902" };
var result:Object = appManager.LaunchApp(params);
var info:String = result.ErrorCode;
```

In addition to `ApplicationId`, there are other optional properties you can define in the parameter object, such as `CmdLine` and `Options`. For `CmdLine` you can pass an argument to the application that it will execute when it launches. You can also define an `Options` object property for the parameter; this contains three properties: `Document`, `Mode`, and `Position`. The `Document` path allows you to define a path for a document you want to open when the application launches, while

www.it-ebooks.info

`Mode` allows you to specify whether the application being launched is an embedded object. The `Position` property allows you to set whether the application should be launched in the `Foreground` or `Background`. For instance, if you wanted to launch the device's Calendar application in the background, you would set the `ApplicationId` to `s60uid://0x10005901`, and the `Position` property to `Background` (see Listing 10-10).



LISTING 10-10: Launching the Calendar application with LaunchApp

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var option:Object = { Position:"Background" };
var params:Object = { ApplicationId:"s60uid://0x10005901",
                      Options:option };

var result:Object = appManager.LaunchApp(params);
var info:String = result.ErrorCode;
```

To launch a new document you need to define the `DocumentPath` property of the document path and then set this on the `Document` property of the parameter object (see Listing 10-11).



LISTING 10-11: Using the LaunchDoc method to open a .txt file

```
import com.nokia.lib.Service;

var appManager:Service = new Service("Service.AppManager", "IAppManager");
var path:Object = { DocumentPath:"c:\\Data\\documentName.txt" };
var params:Object = { Document:path };
var result:Object = appManager.LaunchDoc(params);
var info:String = result.ErrorCode;
```

Next, you'll explore the Calendar Service API.

USING CALENDAR

The Calendar Service API is one of the more complex APIs in the library and requires an understanding of how the actual Calendar service on the mobile device operates, including the data types, structures, and terminology. You can use the Calendar Services API to Access, Create, and Manage Calendar Data on a device.

API Features

The API provides you with extensible functionality to organize events, tasks, and meetings directly on the device, and there are seven methods that you can use in the API:

- `GetList()` — Used to retrieve a list of calendars on a mobile device or a list of calendar entries from a particular calendar

www.it-ebooks.info

- `Add()` — Used to create or update calendars or calendar entries
- `Delete()` — Used to remove a calendar or calendar entry
- `Import()` — Used to import a calendar entry from a file
- `Export()` — Used to export a calendar entry to a file
- `RequestNotification()` — Used to register for an update on a particular calendar entry
- `Cancel()` — Used to cancel an asynchronous method call in the API

In the next few subsections, you'll take a look at the `GetList()`, `Add()`, `Import()`, and `Export()` methods, and cover several features of the Calendar API:

- Locating a calendar on the device
- Understanding calendar entry file formats
- Creating new calendar entries
- Importing and exporting calendar entries

Locating Calendars on a Device

A calendar exists as a single file on the device by default, which can contain any number of calendar entries with various properties, which you'll take a look at shortly.

As shown in Listing 10-12, to use the Calendar API to interact with the Calendar from within a Flash Lite application, you can create a `Service` object instance and call the `Calendar` service provider, using the `IDataSource` interface. You then use the `GetList()` method to retrieve the calendar, specifying the `Type` property as `Calendar`. The `id` property is returned as the `ReturnValue` property.



LISTING 10-12: Using the GetList method to retrieve the id of the calendar on the device

```
import com.nokia.lib.Service;

var calendar:Service = new Service("Service.Calendar", "IDataSource");
var params:Object = { Type:"Calendar" };
var result:Object = calendar.GetList(params);
var calendarId:String = result.ReturnValue;
```

Understanding Calendar Entry File Formats

Before you take a look at creating calendar entries and using the `Import()` and `Export()` methods of the API, consider the file formats used which are saved to the device whenever a new calendar entry is created, iCalendar.

iCalendar (iCal) and vCalendar (vCal)

The iCal and vCal standards are widely used in calendar-based applications, especially in data storage and exchange of entries. The format used by the Calendar API adheres to the Internet

www.it-ebooks.info

Calendaring and Scheduling Core Object Specification (iCal, `http://www.ietf.org/rfc/rfc2445.txt`) and the Electronic Calendaring and Scheduling Exchange Format (vCal, `http://www.imc.org/pdi/vcal-10.txt`).

The following code snippet is an example of a calendar entry in the vCalendar format, specifying the details of an event, where you should be able to make out properties such as start time, `DTSTART`; end time, `DTEND`; and summary, `SUMMARY`:




```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Microsoft Corporation//Windows Calendar 1.0//EN
CALSCALE:GREGORIAN
METHOD:PUBLISH
BEGIN:VEVENT
DTSTART:20100109T000000Z
DTEND:20100109T235959Z
SUMMARY:Roshana's Birthday
UID:C9D63A23-9429-432F-B480-0DA00AEF48DE
END:VEVENT
END:VCALENDAR
```

code snippet birthday.txt

When you import a file into the Calendar using the Calendar Services API, the file needs to be in one of these formats.

Creating a New Calendar Entry

For the API the `CalendarEntry` object contains several properties that provide information about a calendar entry, such as a `Description`, `Summary`, and a range of details that relate to its `StartTime` and `EndTime`.

First, each `CalendarEntry` is defined by its `Type` property, which can be set to one of the following values:

- `Anniversary`
- `DayEvent`
- `Meeting`
- `Reminder`
- `ToDo`

What `Type` the calendar entry is will determine what other properties can be stored or retrieved from `CalendarEntry`. For instance, you can only retrieve information such as: `Attendees`, `Location`, and `Organizer` when `Type` is set to `Meeting`.

You need to use the `Add()` method of the API to create a new `CalendarEntry`. In Listing 10-13 you see that a "to-do" calendar entry object called `entry`, which is created, with the `Type`, `CalendarName`, `Status`, `StartDate`, `EndDate`, `Replication`, `AlarmTime`, and `Description` properties specified. The `Status` property can be set to `TodoInProgress`, `TodoNeedsAction`, or `TodoCompleted`. However, if it is set to `TodoCompleted`, the task will disappear from the calendar.

www.it-ebooks.info



LISTING 10-13: Using the Add method to create a To-Do note entry in a calendar

```
import com.nokia.lib.Service;

var calendar:Service = new Service("Service.Calendar", "IDataSource");

var entry:Object = {  Type:"ToDo",
                      Status:"TodoNeedsAction",
                      StartTime:new Date(),
                      EndTime:new Date(),
                      Replication:"Open",
                      Priority:3,
                      AlarmTime:new Date(),
                      Summary:"Wrox Book",
                      Description:"Read Chapter 10." };

var params:Object = { Type:"CalendarEntry", Item:entry };
var result:Object = calendar.Add(params);
var entryId:String = result.ReturnValue;
var info:String = entryId;
var errorMsg:String = result.ErrorCode.toString();
```

Listing 10-14 shows an Anniversary calendar event for June 14, 2010.



LISTING 10-14: Using the Add method to create an Anniversary entry in a calendar

```
import com.nokia.lib.Service;

var calendar:Service = new Service("Service.Calendar", "IDataSource");

var entry:Object = { Type:"Anniversary",
                     StartTime:new Date(2010, 5, 14, 11, 30, 0, 0),
                     EndTime:new Date(2010, 5, 14, 23, 59, 59, 0),
                     Replication:"Private",
                     Priority:1,
                     AlarmTime:new Date(2010, 5, 14, 11, 25, 0, 0),
                     Summary:"Wedding Anniversary",
                     Description:"Don't forget the flowers.." };

var params:Object = { Type:"CalendarEntry", Item:entry };
var result:Object = calendar.Add(params);
var entryId:String = result.ReturnValue;
var info:String = entryId;
var errorMsg:String = result.ErrorCode.toString();
```

It's possible that a device can have more than one calendar. When you create a `CalendarEntry`, a unique and read-only `Id` property is generated for the instance, which is the "global" `Id` property. In addition to this, a `LocalId` value is generated and this is specific to the calendar the *entry* resides in.

www.it-ebooks.info

Exporting Calendar Entries

Before looking at the `Export()` method let's look at the `Filter` object. As with the other service APIs, you can supply `Filter` as one of the parameters to `GetList()` to retrieve specific calendar information:

- `Filter.DefaultCalendar` — A `Boolean` property that returns the default calendar only. This is the only filter property used when `Type` is set to `Calendar`.
- `Filter.CalendarName` — A string property that specifies the `Calendar` to use.
- `Filter.id` — A string property that matches the `id` property value for a `CalendarEntry` across all calendars on the device.
- `Filter.LocalId` — A string that matches `LocalId` value for a `CalendarEntry`.
- `Filter.StartRange` — A `Date` object used to return `CalendarEntry` objects that start on or after the date specified.
- `Filter.EndRange` — A `Date` object used to return `CalendarEntry` objects that start on or before the date specified.
- `Filter.SearchText` — A string that matches the `Summary` property of the `CalendarEntry`.
- `Filter.Type` — A string property that matches the `Type` of `CalendarEntry`. The `Filter.Type` value can be set to each of the calendar event `Type` values mentioned earlier, and `IncludeAll`, which returns all calendar entries that are on the specified calendar.



Because the `Filter.id` *and* `Filter.LocalId` *properties are unique for each* `CalendarEntry`*, you have to be careful when defining a* `Filter` *object that you don't declare both the* `Id` *and the* `LocalId`*; if either is set, the other* `Filter` *object properties are ignored.*

Shortly you'll see how the filter object can be used to retrieve a specific calendar entry.

The `Export()` method enables you to export entries from a calendar into an iCalendar or vCalendar file. You need to specify the `Type` string and a `Data` object as arguments for the method. The `Type` string just requires the `CalendarEntry` value. The `Data` object is used in a similar way to the `Filter` object; it simply defines the `CalendarEntry` values to be exported.

You specify the `CalendarName` to define the exact calendar to use, and a `FileName` property to define the full file name and path of the exported file. The `Format` property lets you determine whether the file is in the iCalendar or vCalendar format by specifying either "ICal" or "VCal". You also have the option to define which `CalendarEntry` values are exported from the `Id` or `LocalId` values it possesses, using either `IdList` or `LocalIdList`.

The Anniversary entry created earlier could be searched for via its `Summary` property. Listing 10-15 shows how `Filter.SearchText` helps to retrieve a calendar entry with `Anniversary` in the Summary via `GetList()`. The results are then added to the `idList` array and exported to a file.

www.it-ebooks.info



LISTING 10-15: Defining the SearchText property to filter and export a calendar entry to a file

```
import com.nokia.lib.Service;

var calendar:Service = new Service("Service.Calendar", "IDataSource");
var filter:Object = { SearchText:"Anniversary" };
var params:Object = { Type:"CalendarEntry", Filter: filter };
var result:Object = calendar.GetList(params);
var entryList:Object = result.ReturnValue;
var calendarEntry:Object = null;
var idList:Array = [];

do {

      calendarEntry = entryList.next();

      if (null != calendarEntry) {

            idList.push(calendarEntry.id);

      } else {

            break;
      }

} while (true);

var data:Object = { IdList:idList,
                    Format:"VCal",
                    FileName:"C:\\Data\\anniversary.txt" };

params = { Type:"CalendarEntry", Data:data };
result = calendar.Export(params);
```

Next, take a look at importing calendar entries.

Importing Calendar Entries

Use the `Import()` method to read entries into a calendar. The method requires you to specify two properties for the main argument: the `Type`, which is set to `CalendarEntry`, and a `Data` object property similar to the object used for the `Export()` method. In Listing 10-16 you'll see that the `Data` object has two properties defined. As with the `Export()` method, the `Format` property allows you to specify the data format of the files as either "ICal" or "VCal". This obviously needs to match up with the actual file format. The `FileName` property is also set to determine the full path to the file that is to be imported; here, the `birthday.txt` file from the earlier code snippet is used. You can also specify the `CalendarName` to match the name of the calendar you want to import `CalendarEntry` objects into, and if the default calendar does not exist, it is created.

www.it-ebooks.info



LISTING 10-16: Importing calendar entries into the calendar

```
import com.nokia.lib.Service;

var calendar:Service = new Service("Service.Calendar", "IDataSource");
var data:Object = { Format:"VCal", FileName:"C:\\Data\\birthday.txt" };
var params:Object = { Type:"CalendarEntry", Data:data };
result = calendar.Import(params);
var info:String = result.ReturnValue;
var errorMsg:String = result.ErrorCode.toString();
```

Next, you'll explore the Contacts Service API.

USING CONTACTS

The Contacts Service API ultimately allows you to connect with an address book and interact with contacts' data on the device. You can use the API to access and manage information about a user's contacts.

API Features

There are seven methods you can utilize in the Contacts Services API:

- `Add()` — Used to add new contacts and groups or update existing ones
- `Delete()` — Used to delete contacts or contact groups from a database
- `Export()` — Used to export a contact to file
- `GetList()` — Used to retrieve a list of contacts
- `Import()` — Used to import a contact from file
- `Organise()` — Used to organize contacts into groups
- `Cancel()` — Used to cancel an asynchronous Contacts API method

Over the next few subsections, you'll take a look at the `GetList()`, `Add()`, `Import()`, and `Export()` methods in depth, covering several features of the Contacts API, including:

- Retrieving contacts, contact groups, and databases
- Handling results from the `GetList()` call
- Adding new contact and contact group information
- Updating existing contact and contact group information
- Removing existing contacts or groups
- Importing and exporting vCard files
- Organizing contacts into groups

www.it-ebooks.info

Retrieving Contacts, Contact Groups, and Databases

Contacts are stored in databases, which can exist as a file on a device file system or as a SIM card entity. Contact databases are referenced by their URI values, which define their location. For instance the URI value for the default contacts database found on a Nokia N97 is `cntdb://c:contacts.cdb`.

The `GetList()` method allows you to get information about contacts, whether it's the database URI, a specific contact, or a group. You can specify the `Type` property as one of the following values:

- `Contact` — Used to return a list of contacts
- `Database` — Used to return a list of contact databases
- `Group` — Used to return a list of contact groups.

Listing 10-17 shows the `GetList()` method for retrieving contacts on a device. The value returned for the contact's `FirstName` and `LastName` will be discussed shortly.


Available for download on Wrox.com

LISTING 10-17: Listing contacts using GetList

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var params:Object = { Type:"Contact" };
var result:Object = contacts.GetList(params);
var contactList:Object = result.ReturnValue;
var contactEntry:Object = null;

do {
      contactEntry = contactList.next();

      if (null != contactEntry) {

            var firstName:String = contactEntry.FirstName["Value"];
            var lastName:String = contactEntry.LastName["Value"];

      } else {

            break;

      }

} while (true);

var errorId:Number = result.ErrorCode;
```

The API also provides you with a way to set the criteria for the information retrieved using the `Filter` object, which you define to retrieve information about contacts or contact groups. For retrieving database information you don't define a `Filter` object.

www.it-ebooks.info

The `Filter` object has three properties which can be set:

- `DBUri` — A string property representing the URI of the database
- `id` — A string property that represents the unique identifier for a contact or group
- `SearchVal` — A string property that can be used with a `Contact` search only. The text string defines the return results, which should be based on a contact's first or last name.

In addition to `Filter`, there is the `Sort` object, which you can provide to define how your `Contact` search information is returned. The `Sort` object just has one property, called `Order`, where you have the option of specifying whether information should be `Descending` or `Ascending`.

In Listing 10-18 the `Filter` property `SearchVal` is defined to return contacts with either a first name or last name of "Coombes", and the `Sort` property `Order` is set to `Descending`.



LISTING 10-18: Filtering contacts using the SearchVal property

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var filter:Object = { SearchVal:"Coombes" };
var sort:Object = { Order:"Descending" };
var params:Object = { Filter:filter, Sort:sort, Type:"Contact" };
var result:Object = contacts.GetList(params);
var contactList:Object = result.ReturnValue;
var contactEntry:Object = null;

do {
      contactEntry = contactList.next();

      if (null != contactEntry) {

            var firstName:String = contactEntry.FirstName["Value"];
            var lastName:String = contactEntry.LastName["Value"];

      } else {

            break;

      }

} while (true);

var errorId:Number = result.ErrorCode;
```

Handling Results from the GetList() Call

For each contact returned by `GetList()`, there are two main types of properties that you need to get acquainted with, an `id` property and a series of object properties that represent contact information, called a `Key`.

www.it-ebooks.info

The `id` property is simply a unique value for a contact, which can be referenced when updating contact information via the `Add()` method. However, the `key` is much more complex. The `key` represents a piece of information about a contact. As you've already seen, for example, the `FirstName` and `LastName`, and other properties, such as `EmailHome`, `EmailWork`, and `JobTitle`, are examples of `key` names that you can define or retrieve using the API.

A `key` contains the following properties:

- `Label` — A string that describes the key value
- `Value` — A string that defines the key
- `Next` — An object, which can contain an unlimited number of nested keys

These represent what is referred to as the `Label` for the `key` and associated with each `Label` is the value. To retrieve the value for the contact `Label`, you have to assign `Value` to the `Label` name (see Listing 10-19).


Available for download on Wrox.com

LISTING 10-19: Retrieving contact key values

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var params:Object = { Type:"Contact" };
var result:Object = contacts.GetList(params);
var contactList:Object = result.ReturnValue;
var contactEntry:Object = null;

do {
      contactEntry = contactList.next();

      if (null != contactEntry) {

            var firstName:String = contactEntry.FirstName["Value"];
            var lastName:String = contactEntry.LastName["Value"];
            var prefix:String = contactEntry.Prefix["Value"];
            var emailHome:String = contactEntry.EmailHome["Value"];
            var phoneHome:String = contactEntry.LandPhoneHome["Value"];
            var voipHome:String = contactEntry.VoipHome["Value"];
            var mobileHome:String = contactEntry.MobilePhoneHome["Value"];
            var ringtone:String = contactEntry.Ringtone["Value"];
            var urlHome:String = contactEntry.URLHome["Value"];

      } else {

            break;

      }

} while (true);
```

www.it-ebooks.info

When using `GetList()` to retrieve database information, the `ReturnValue` simply returns the `DBUri` string property, which represents the URI of the contact database.

When using `GetList()` to return information about contact groups, the `ReturnValue` consists of three parameters:

- `id` — A string property representing the unique identifier
- `GroupLabel` — A string property representing the name of the group
- `Contents` — An array containing all the contact `Id`s assigned to the group

Listing 10-20 shows how group information can be retrieved. When trying this out on the device, you need to ensure that there are groups created on the device and that there are members in those groups.



LISTING 10-20: Retrieving contact groups' properties

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var params:Object = { Type:"Group" };
var result:Object = contacts.GetList(params);
var groupList:Object = result.ReturnValue;
var groupEntry:Object = null;

do {
      groupEntry = groupList.next();

      if (null != groupEntry) {

            var groupId:String = groupEntry.id;
            var groupContacts:String = groupEntry.Contents;
            var groupLabel:String = groupEntry.GroupLabel;

      } else {

            break;

      }

} while (true);
```

Adding New Contact and Contact Group Information

You can use the `Add()` method to create new contact information. You need to define two properties on the parameters object: `Type` and `Data`. For a new contact, you need to set the type to `Contact`, and for a new group you set this to `Group`. For new contacts you also need to define the `key` values for the `Data` object, while for groups you simply define the `GroupLabel` property. When adding either a new contact or a new group, you have option of specifying the database `DBUri` to

www.it-ebooks.info

which to add the information to. Alternatively, leaving this property out of the definition will simply add the new information to the default database on the device.

In Listing 10-21 a new contact with just the `FirstName` property defined for data is created.



LISTING 10-21: Creating a new contact with the name "Jermajesy"

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var contactName:Object = { Label:"Name", Value:"Jermajesty" };
var data:Object = { FirstName:contactName };
var params:Object = { Type:"Contact", Data:data };
var result:Object = contacts.Add(params);
```

In Listing 10-22 you'll see how you can create a new contact group with the label `Wrox`.



LISTING 10-22: Creating a new contact group labelled "Wrox"

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var data:Object = { GroupLabel:"Wrox" };
var params:Object = { Type:"Group", Data:data };
var result:Object = contacts.Add(params);
```

To update an existing contact or group in a database, you also use the `Add()` method. To do this, simply specify the `Type` property in the `params` object as either `Contact` or `Group`. Then for the `Data` object property, you need to specify values that define your contact or group, supplying the `id` of the entity you want to update.

Importing vCard Files

The `Import()` method provides you with an alternative way to add new contacts to a database from a source file in the vCard format. Here's a very basic example of the vCard 3.0 format:




```
BEGIN:VCARD
VERSION:3.0
FN:Jermaine Anderson
END:VCARD
```

code snippet jermaine.vcf

A new `Data` object property to note here is the `Data.SourceFile`. This is the full path to the vCard on the file system for the device. The file can be imported into contacts, using the `Import()` method (see Listing 10-23).

www.it-ebooks.info



LISTING 10-23: Importing a contact from a file on the device

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var data:Object = { SourceFile:"C:\\Data\\jermaine.vcf" };
var params:Object = { Type:"Contact", Data:data };
var result:Object = contacts.Import(params);
```

Exporting vCard Files

To export a contact to a file, you can use the `Export()` method of the Contacts API. The arguments for the method require you to define a `Data` object that contains the `DestinationFile` property and `id` property of the parameter for the `Contact` you want to export to the file system.

In Listing 10-24 the `contactId` defined should be retrieved from the actual ID of the contact you want to export. See if you can export the contact created earlier. *Hint*: you would need to use `GetList()` and a `Filter` object.



LISTING 10-24: Exporting a contact to a file on the device

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var data:Object = { DestinationFile:"C:\\Data\\jermajesty.vcf", id:contactId };
var params:Object = { Type:"Contact", Data:data };
var result:Object = contacts.Export(params);
```

Organizing Contacts in Groups

You can arrange the grouping of contacts in a database through two values, one of which you must specify in the `OperationType` property. This can be set to one of the following values:

- `Associate` — Adds a contact to a group
- `Dissociate` — Removes a contact from a group.

When using the `organize()` method, you need to bear in mind the type of operation you want to use. Create an array containing the `id` values for the contacts you want to arrange and assign it to the `Data.IdList` property. You also need to specify the `Type` parameter as `Group`.

In Listing 10-25 the `contactId` specified for the contacts array needs to be retrieved from the actual ID of the contact who needs to be associated with a group. In addition, the `groupId` specified for the `id` property of the `Data` object needs to be retrieved. Have a go at associating a contact by retrieving these pieces of information first. Again, you should be able to retrieve the missing ID value by using the `GetList()` method.

www.it-ebooks.info



LISTING 10-25: Associating a contact with a contact group

```
import com.nokia.lib.Service;

var contacts:Service = new Service("Service.Contact", "IDataSource");
var contactsList:Array = [contactId];
var data:Object = { id:groupId, IdList:contactsList };

var params:Object = { Type:"Group",
                      Data:data,
                      OperationType:"Associate" };

var result:Object = contacts.Organise(params);
```

Next, you take a look at Landmarks.

USING LANDMARKS

The Landmarks Services API allows you to consume, manage, and distribute location information as part of your Flash Lite applications. A good use case for Landmarks would be a travel guide application, where users are able to download and import a tourist destination as a Landmark onto their device.

API Features

There are eight methods that you can utilize in the Landmarks Services API:

- `Add()` — Used to create new or update Landmarks and Landmark categories
- `Delete()` — Used to delete a Landmark or Landmark category from the device
- `Export()` — Used to export a Landmark to a file
- `GetList()` — Used to retrieve a list of Landmarks
- `Import()` — Used to import a Landmark file
- `New()` — Used to create empty Landmarks or Landmark categories
- `Organise()` — Used to organize Landmarks into categories
- `Cancel()` — Used to cancel asynchronous calls

In the next few subsections you'll take a look at the `GetList()`, `Add()`, `Import()`, and `Export()` methods, and cover several functional aspects of the Landmark API:

- Creating Landmarks
- Adding a new Landmark
- Creating Landmark categories

Before moving onto these methods, take a look at the properties of a Landmark used as part of the API.

www.it-ebooks.info

Landmarks

A Landmark object can contain the following information:

- `Name` — The name of the Landmark
- `Description` — A textual description of the Landmark
- `Category` — The name of the category, for example, Hotel
- `Coordinates` — The data coordinates of the Landmark (e.g., latitude and longitude) with accuracy information
- `Address` — The address information of the Landmark
- `Coverage radius` — The radius of the area covered by the Landmark

The Landmark format is defined in the S60 Platform Landmarks Exchange Format Specification 1.0. Because the specification is open, the format can be used across multiple devices that support it. Hence, an end user is able to export and import Landmarks, and send them to other devices.

Similarly to the Calendar and Contacts, there is also a default Landmark database on the device, which can be found at `file://c:eposlm.ldb`. If you don't specify which database you want to operate the API calls on, you will always default to this database.

Creating New Landmarks

There are two ways to create a Landmark. You can use the `New()` method followed by `Add()`, editing existing Landmark detail by referencing the Landmark's ID property, `Id`. Alternately, you can use the `Add()` method in isolation, creating the detail for a Landmark without specifying an `Id`. First, use `New()` to obtain the `Id` property of the newly created Landmark. Specify the parameter object's `Type` value as `Landmark` (see Listing 10-26).



LISTING 10-26: Creating a new Landmark template

```
import com.nokia.lib.Service;

var landmark:Service = new Service("Service.Landmarks", "IDataSource");
var params:Object = { Type:"Landmark" };
var result:Object = landmark.New(params);
var landmarkId:Number = result.ReturnValue.Id;
```

You can then assign the rest of the detail for the Landmark to the parameter object's `Data` property, which needs to be passed to the `Add()` method. This creates the new Landmark in the default Landmark database.

Listing 10-27 demonstrates how you can edit an existing Landmark that has no data assigned.

www.it-ebooks.info



LISTING 10-27: Editing a pre-existing Landmark

```
import com.nokia.lib.Service;

var landmark:Service = new Service("Service.Landmarks", "IDataSource");

var params:Object = { Type:"Landmark" };
var result:Object = landmark.New(params);

var landmarkId = result.ReturnValue.id;

var data:Object = { Id:landmarkId,
                    LandmarkName:"Crete, Almiryda",
                    LandmarkDesc:"Joey's Summer destination" };

var params:Object = { Type:"Landmark",
                      Data: data };

var result:Object = landmark.Add(params);
```

For a more detailed Landmark creation, you need to specify the coordinates `Latitude` and `Longitude`, which are properties of the `LandmarkPosition` object property set on `LandmarkData` (see Listing 10-28).



LISTING 10-28: Specifying the coordinates of a Landmark

```
import com.nokia.lib.Service;
var landmark:Service = new Service("Service.Landmarks", "IDataSource");

var params:Object = { Type:"Landmark" };
var result:Object = landmark.New(params);

var landmarkId = result.ReturnValue.id;

var coordinates:Object = { Latitude:35.4491940,
                           Longitude: 24.1981300 };

var data:Object = { Id:landmarkId,
                    LandmarkName:"Crete, Almiryda",
                    LandmarkDesc:"Joey's Summer destination",
                    LandmarkPosition:coordinates };

var params:Object = { Type:"Landmark",
                      Data: data };

result = landmark.Add(params);
```

In the preceding example, the `LandmarkPosition.Latitude` and `LandmarkPosition.Longitude` for the coordinates of Crete are defined in `landmarkPos`.

www.it-ebooks.info



To create a Landmark or Landmark category using the `Add()` *method alone you do not need to specify the* `Id` *property of the* `Data` *object, as the system will generate the new* `Id` *automatically. When editing an existing Landmark, you need to retrieve the Landmark's* `Id` *through the* `getList()` *method, and then use the* `Add()` *method, specifying this value.*

You can also specify the `Data.DatabaseURI` property to point to a specific Landmark database for the operation.

Here, you use the `Data` property of the `params` object to specify information for your Landmark. The Landmark can have the following information defined:

- `LandmarkName` — A string property name given to the Landmark, which can be anything, up to a maximum length of 255 characters.
- `LandmarkDesc` — A string property description given to the Landmark, which can be anything, up to a maximum length of 4095 characters.
- `Id` — A string property that defines the unique ID of the Landmark. The ID is unique only within the database to which the Landmark belongs.
- `CategoryInfo` — An array property that contains string values corresponding to the ID of one or more Landmark Category IDs.
- `CoverageRadius` — A number property that indicates the area covered by the Landmark. The measurement for the radius is in meters from the central `LandmarkPosition`.
- `IconFile` — A string defining the full path and file name to the `.mbm` icon file that contains the icon associated with the Landmark.
- `IconIndex` — A number that specifies the index of the icon within the icon file.
- `IconMaskIndex` — A number that specifies the index of the icon mask within the icon file.
- `LandmarkPosition` — This is an object containing number properties that specify the location of the Landmark including: `Latitude`, `Longitude`, and `Altitude`. The `HAccuracy` and `VAccuracy` properties give an error estimate for horizontal and vertical alignment in relation to the coordinates, respectively.
- `LandmarkFields` — An object containing string properties with more detail on the Landmark. These include the street name, `Street`; building name, `BuildingName`; `District`; `City`; `area` code, `AreaCode`; contact number; `Telephone`; and `Country`.

Individual Landmarks exist as files on the device in the `.lmx` file format. The following is an example snippet of Landmark `.lmx` output.




```
<?xml version="1.0" encoding="UTF-8"?>
<lm:lmx xmlns:lm="http://www.nokia.com/schemas/location/landmarks/1/0" xmlns:
xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
www.nokia.com/schemas/location/landmarks/1/0/ lmx.xsd">
      <lm:landmarkCollection>
            <lm:landmark>
```

www.it-ebooks.info

```
                <lm:name>Crete, Almiryda</lm:name>
                <lm:description> Joey's Summer destination</lm:description>
                <lm:coordinates>
                      <lm:latitude>35.4491940</lm:latitude>
                      <lm:longitude>24.1981300</lm:longitude>
                      <lm:horizontalAccuracy>1727</lm:horizontalAccuracy>
                </lm:coordinates>
                <lm:addressInfo>
                      <lm:street></lm:street>
                      <lm:postalCode></lm:postalCode>
                      <lm:city></lm:city>
                      <lm:state></lm:state>
                      <lm:country></lm:country>
                      <lm:phoneNumber></lm:phoneNumber>
                </lm:addressInfo>
                <lm:category>
                      <lm:id>33000</lm:id>
                      <lm:name>Places of worship</lm:name>
                </lm:category>
          </lm:landmark>
    </lm:landmarkCollection>
</lm:lmx>
```

code snippet almyrida.lmx

Importing and Exporting Landmarks

To import a Landmark from a file you use the `Import()` method. In Listing 10-29 you see the `SourceFile` and `MimeType` properties defined for the `Data` property of the parameters object. The `Type` property of `params` object then needs to be set to `Landmark` before it is supplied to the `Import()` method.



LISTING 10-29: Importing an .lmx file into a Landmark database

```
import com.nokia.lib.Service;

var landmark:Service = new Service("Service.Landmarks", "IDataSource");

var data:Object = { SourceFile:"C:\\Data\almyrida.lmx",
                    MimeType:"application/vnd.nokia.landmarkcollection+xml" };

var params:Object = { Type:"Landmark", Data:data };
var result:Object = landmark.Import(params);
var landmarkList:Object = result.ReturnValue;
var item:Object = null;

do {
      item = landmarkList.next();

      if (null != item) {
```

www.it-ebooks.info

```
            var landmarkId = item.id;

        } else {
            break;
        }

    } while (true);
```

To export a Landmark from a file, you use the `Export()` method. In Listing 10-30 you see the `DestinationFile` and `MimeType` properties defined for the `Data` property of the parameters object. The "id1" value, which is defined for the `idList` array, needs to be replaced with the ID of the Landmark you want to export, and the name of the file `OutputFile.lmx` needs to be replaced with the chosen file name.



LISTING 10-30: Using the Landmark Export method to export an .lmx file

```
import com.nokia.lib.Service;

var landmark:Service = new Service("Service.Landmarks", "IDataSource");
var idList:Array = ["id1"];

var data:Object = { IdList:idList,
                    DestinationFile:"C:\\Data\OutputFile.lmx",
                    MimeType:"application/vnd.nokia.landmarkcollection+xml" };

var params:Object = { Type:"Landmark", Data:inputData };
var result:Object = landmark.Export(params);
```

Creating Landmark Categories

Landmark categories have only a few properties, including the `CategoryName`; the unique `id` of the category is only unique for a particular database in that another database's category could have the same value. There is also the `GlobalId`, which is a unique `Id` given to a Landmark category across all Landmark databases.

USING LOCATION

The Location Services API gives Flash Lite developers access to the services on Nokia Devices that utilize Global Positioning System (GPS). Using the Location Service API you can retrieve the coordinates of a device. The next chapter utilizes the Location Service API as part of the build for the Weather client application.

API Features

There are four methods of the Location Service API in particular that you use for estimating a device's global position and calculating distances:

- `GetLocation()` — Used to retrieve the coordinates of the device

www.it-ebooks.info

- `Trace()` — Used to retrieve the coordinates of the device using a defined time interval
- `CancelNotification()` — Used to cancel asynchronous method calls
- `Calculate()` — Used to make calculations based on device positions

In the next few subsections, you'll take a look at the `GetLocation()`, `Trace()`, `Calculate()`, and `CancelNotification()` methods, and cover the following functional aspects of the Location API:

- Retrieving the location of a device
- Retrieving location data automatically
- Canceling calls to retrieve a location
- Measuring distances

Retrieving the Location of a Device

The `GetLocation()` method can be used synchronously or asynchronously to retrieve the location of a device. Again, you need to specify a parameterized object for the method, in which you define the type of location information and *how* it is returned.

First, there's the `LocationInformationClass` property, which allows you to specify the level of location detail; this can be set to one of the following string values:

- `BasicLocationInformation` — Used to retrieve basic information in the `ReturnValue` object, including `Longitude`, `Latitude`, and `Altitude` estimates
- `GenericLocationInfo` — Used to retrieve a wider range of information, which includes details such as number of satellites used to retrieve the information

Listing 10-31 demonstrates how to retrieve the `Altitude`, `Longitude`, and `Latitude` values from `GetLocation()`. After specifying a parameter object and calling the `GetLocation()` method, supplying either the `BasicLocationInformation` or `GenericLocationInfo` will return a `ReturnValue` object with the following properties:

- `Longitude` — A value between +180 and -180, representing estimated degrees
- `Latitude` — A value between the +90 and -90, representing estimated degrees
- `Altitude` — A value between +90 and -90, representing an elevation estimate, in meters

For any property in the API requiring use of these properties these values apply.



LISTING 10-31: Using the GetLocation method to retrieve data coordinates

```
import com.nokia.lib.Service;

var location:Service = new Service("Service.Location", "ILocation");
var params:Object = { LocationInformationClass:"BasicLocationInformation" };
var result:Object = location.GetLocation(params);
var coordinates:Object = result.ReturnValue;
var altitude:Number = coordinates.Altitude;
var longitude:Number = coordinates.Longitude;
```

www.it-ebooks.info

Retrieving Location Data Automatically

The asynchronous `Trace()` method allows you to retrieve "basic" and "generic" information automatically. For this, you need to define an additional property for the parameter object, called `Updateoptions`. You define the `Updateoptions` object when you want to manage the frequency of the updates of the locations or if you want to *ensure* that particular information is retrieved. The object has four properties; all are optional:

- `Updateoptions.UpdateTimeOut` — A number representing the duration in microseconds for how long to wait for location information to be retrieved. The value must be greater than `UpdateMaxAge` and less than the `UpdateInterval`. The default for the value is "15000000" microseconds, which is equal to 15 seconds.
- `Updateoptions.UpdateInterval` — A number representing the lag between two consecutive location estimates. The value is in microseconds and must be greater than `UpdateMaxAge` and less than the `UpdateTimeOut` value. The default for the value is "1000000" microseconds, which is equal to 1 second.
- `Updateoptions.UpdateMaxAge` — A number specifying the expiration time for the cache location information. The default for this value is 0 microseconds, and it must be less than both `UpdateInterval` and `UpdateTimeOut` values.
- `Updateoptions.PartialUpdates` — A `Boolean` used to specify whether the `ReturnValue` retrieved contains at least "basic" information in line with the `false` setting. When set to `true`, however, no specific information is guaranteed to be retrieved.

Listing 10-32 shows how to implement the `Trace()` method, setting the `UpdateInterval` property to 5 seconds.


Available for download on Wrox.com

LISTING 10-32: Using the Trace method to retrieve data coordinates automatically

```
import com.nokia.lib.Service;

var location:Service = new Service("Service.Location", "ILocation");

var options:Object = { UpdateInterval:5000000 };

var params:Object = { LocationInformationClass:"GenericLocationInfo",
                      Updateoptions:options };

location.Trace(params, onUpdate);

function onUpdate(transactionID:Number, eventID:String, result:Object) {

      if (result.ErrorCode == 0) {

            var coordinates:Object = result.ReturnValue;

            var altitude:Number = coordinates.Altitude;
            var longitude:Number = coordinates.Longitude;
            var latitude:Number = coordinates.Latitude;
```

continues

www.it-ebooks.info

LISTING 10-32 *(continued)*

```
        } else {

              var errorId:Object = result.ErrorCode;
        }
}
```

Canceling Calls to Retrieve a Location

The `CancelNotification()` is a synchronous method you can call on the location `Service` instance, which as the name suggests, allows you to cancel asynchronous calls for the Location Service API. This requires the `CancelRequestType` property to be specified as a parameter (see Listing 10-33). To cancel a `GetLocation()` call, you specify `GetLocCancel`, whereas to cancel `Trace()` you specify `TraceCancel`.



LISTING 10-33: Using the CancelNotification method to cancel an asynchronous method in the Location API

```
import com.nokia.lib.Service;

var location:Service = new Service("Service.Location", "ILocation");
var params:Object = { CancelRequestType:"TraceCancel" };
var result:Object = location.CancelNotification(params);
```

Measuring Distances

You can use the synchronous `Calculate()` method to determine the coordinates, distance between, and bearings of two location points. There are a total of five properties that can be defined as part of the parameters object for the method:

- `MathRequest` — Used to define the type of calculation to perform
- `DistanceParamSource` — Used to specify the coordinates of the start location
- `DistanceParamDestination` — Used to specify the coordinates of the end location
- `MoveByThisDistance` — Used to specify a new distance
- `MoveByThisBearing` — Used to specify a new bearing

The `Calculate()` method requires the `MathRequest` and `DistanceParamSource` properties are specified. You define `MathRequest` to specify the type of calculation. You can set this to the following values:

- `FindDistance` — Used to obtain a distance in meters
- `FindBearingTo` — Used to obtain the bearing measured in degrees from true north, clockwise
- `MoveCoordinates` — Used to obtain a new location object

www.it-ebooks.info

You define the `Longitude`, `Latitude`, and `Altitude` value properties in the `DistanceParamSource` object, which represents the starting position for a location. In order to calculate coordinates, you need to specify `DistanceParamDestination`, representing the target location as an optional parameter for the `Calculate()` method. As with the `DistanceParamSource` you specify the `Longitude`, `Latitude`, and `Altitude` values.

Listing 10-34 demonstrates how to implement the `Calcaluate()` method, calculating the distance between `home` and `stadium`.



LISTING 10-34: Calculating distances between two locations

```
import com.nokia.lib.Service;

var location:Service = new Service("Service.Location", "ILocation");

var home:Object = { Longitude:-1.417782,
                    Latitude:53.421129,
                    Altitude:0.5 };

var stadium:Object = { Longitude:-1.501856,
                       Latitude:53.411712,
                       Altitude:0.5 };

var params:Object = { MathRequest:"FindDistance",
                      DistanceParamSource:home,
                      DistanceParamDestination:stadium };

var result:Object = location.Calculate(params);

var distance:Object = result.ReturnValue;
```

The result of the calculation returns 5669.29, which represents the distance calculated in meters.

When you specify the `MoveCoordinates` value in `MathRequest`, you need to define the following properties on the argument to the `Calculate()` method:

- `MoveByThisDistance` — A value representing the distance value in meters
- `MoveByThisBearing` — A value representing a new bearing value measured in degrees from true north, counting clockwise

Next, take a look at the Messaging Service API.

USING MESSAGING

You use the Messaging Services API to manage, monitor, and send messages. In this section you go through the Messaging Services API of S60, which gives you the ability to read the inbox of a Nokia device from within your Flash Lite applications.

www.it-ebooks.info

There are various methods in the API that you will be able to use to allow you to list, read, write, send, and delete messages on a device. In addition, you can create notifications and monitor an inbox, so that, when a message is received, you can gracefully handle any interruptions or notify your users during the use of an application.

API Features

There are seven methods currently available in the Messaging Services API:

- `GetList()` — Used to retrieve a list of messages from the inbox
- `Send()` — Used to send messages from the inbox
- `RegisterNotification()` — Used to register for notifications when messages are received by the inbox
- `CancelNotification()` — Used to cancel notifications
- `ChangeStatus()` — Used to change the status property of a message
- `Delete()` — Used to delete a message from the inbox
- `Cancel()` — Used to cancel an asynchronous method.

In the following subsections, you'll cover the `GetList()`, `Send()`, `RegisterNotification()`, `CancelNotification()`, `ChangeStatus()`, and `Delete()` methods, covering the following aspects of using the Messaging API:

- Retrieving messages from an inbox
- Message attachments
- Filtering inbox messages
- Sorting inbox messages
- Sending messages
- Monitoring an inbox for incoming messages
- Modifying the status of a message
- Removing messages from the inbox

Before you move onto these, let's take a look at the features of message items.

Messages

Any message that resides in the inbox of a device has numerous properties, some of which you may already be familiar with:

- `Sender` — Where the message came from. Usually a mobile number, a service or a contact name from the contacts list on a device.
- `Subject` — A description or a title for the message, or partial message content
- `Time` — The date and time for when a message was received
- `BodyText` — The content for the message

www.it-ebooks.info

There are additional properties you can retrieve from a single message item using the Messaging Services API. For instance, you can retrieve and reference specific message's through the `MessageId` property, and also use the `MessageType` to determine whether a message is "SMS," "MMS," or "unknown." The `MessageId` property is a unique identifier for a message on the device and is generated automatically.

You can also determine how important a message is via the `Priority` property, which can be set to `Low`, `Medium`, or `High`. The `Unread` property gives an indication of whether a message has been read or not. Each message item can also have the standard recipient properties defined: `To`, `Cc`, and `Bcc`, each of which contains an array of strings.

Messages can also have attachments, which we'll discuss in more detail shortly. First, let's have a look at the way to retrieve messages, using the `GetList()` method.

Retrieving Messages from an Inbox

To list the contents of an inbox and retrieve messages, you use the `GetList()` method of a `Service` object instantiated with `Service.Messaging`, and the `IMessaging` interface. You need to set the `params` object `Type` property to `Inbox`, and as with the other services, you need to create a `Service` object instance and then call the Messaging Service provider using the `IMessaging` interface.

Listing 10-35 demonstrates how to retrieve messages from an inbox. As mentioned earlier the `result` object assigned to the `GetList()` method should return `ErrorCode`, `ErrorMessage`, and `ReturnValue` properties. The `result.ReturnValue` should contain an array of message object items from the inbox. These are assigned to the `msg` object.



LISTING 10-35: Retrieving messages from an inbox

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");
var params:Object = { Type:"Inbox" };
var result:Object = messaging.GetList(params);
var messages:Array = result.ReturnValue;
var info:String = "Messages" + newline;

if(result.ErrorCode == 0){

      var msg:Object;
      var sender:String;
      var subject:String;
      var messageId:String;
      var messageType:String;
      var time:String;

      do {
            msg = messages.next();
```

continues

www.it-ebooks.info

LISTING 10-35 *(continued)*

```
            if (null != msg) {

                  sender = msg.Sender;
                  subject = msg.Subject;
                  messageId = msg.MessageId;
                  messageType = msg.MessageType;
                  time = msg.Time;

                  info += sender + " | " + subject + " | " + messageId + newline;
                  info += messageType + " | " + time + newline;

            } else {

                  info += "Total Messages = " + messages.length;
                  break;

            }

      } while (true);
}
```

In the example shown, you can see that reading each `msg` property is straightforward. First, a check is made to see whether an error has occurred in the service call via the `ErrorCode` property, where the value 0 denotes "no error." Then a `do-while` statement is used to loop through each `msg` in the `ReturnValue` array property, which is assigned to the inbox.

Message Attachments

A message could potentially have any number of attachments. The `Attachment` property is what determines whether a message item contains an attachment, and in the previous example for `msgObj`, this would be set to `true`. This property can then be used to determine if it is possible to read the `AttachmentList` array, which should contain a number of attachment objects. Continuing on from the last example, the following snippet demonstrates how you can detect message attachments, using the variable `attachmentObj`.

Attachment objects contain several properties: `FileName`, `FileHandle`, `MimeType`, and `FileSize` (see Listing 10-36).



LISTING 10-36: Using the GetList method to list message attachments in the inbox

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");
var params:Object = { Type:"Inbox" };
var result:Object = messaging.GetList(params);
var messages:Array = result.ReturnValue;

var info:String = "Messages" + newline;
```

www.it-ebooks.info

```
if(result.ErrorCode == 0){

      var msg:Object;
      var sender:String;
      var subject:String;
      var messageId:String;
      var messageType:String;
      var time:String;
      var fileName:String;
      var fileSize:String;
      var mimeType:String;
      var fileHandle:String;

      do {
            msg = messages.next();

            if (null != msg) {

                  sender = msg.Sender;
                  subject = msg.Subject;
                  messageId = msg.MessageId;
                  messageType = msg.MessageType;
                  time = msg.Time;

                  info += sender + " | " + subject + " | " + messageId + newline;
                  info += messageType + " | " + time + newline;

                  var attachment:Object;

                  if (msg.Attachment){

                        for(var i:Number = 0; i< msg.AttachmentList.length; i++ ){

                              attachment = msg.AttachmentList[i];

                              fileName = attachment.FileName;
                              fileSize = attachment.FileSize;
                              mimeType = attachment.MimeType;
                              fileHandle = attachment.Filehandle;

                              info += fileName;
                              info += fileSize;
                              info += mimeType;
                              info += fileHandle;
                        }
                  }

            } else {

                  info += "Total Messages = " + messages.length;
                  break;

            }

      } while (true);
}
```

www.it-ebooks.info

Filtering Inbox Messages

The `GetList()` method for messaging also accepts an optional `Filter` object, as a parameter, which has a number of properties you can predefine for obtaining the messages you want to retrieve from the Inbox on a device:

- `MessageTypeList` — If you only want to retrieve a particular type of message from an inbox, you provide the `MessageTypeList` as `SMS`, `MMS`, or both, in an array.
- `MessageId` — If you know the exact ID of a message, you can set the `MessageId` property to retrieve the message with that ID. This is useful in scenarios where you have already used `GetList()` to retrieve messages from your inbox.
- `SenderList` — If you want to retrieve messages from the inbox based on who sent a message, you can specify the `SenderList` array property. To obtain messages based on one or more senders, you need to specify each sender name in the array as a string.
- `Subject` — You can use this if you want to retrieve messages from the inbox based on the subject heading. However, this only applies to messages with `MessageType` set to `MMS`; hence, specifying this value will not return any SMS messages.
- `StartDate` and `EndDate` — If you want to retrieve messages based on when they were sent to an inbox, you define the `StartDate` and `EndDate`. These properties are both `Date` objects, which can be used together or in isolation. If you specify both properties in the `Filter` object, you will retrieve messages in your inbox in the date range specified, providing that the `EndDate` is after the `StartDate`. If the `EndDate` is specified alone, then the criterion is for messages returned on and before the date, whereas for the `StartDate` the messages returned will be on and after the date.

Listing 10-37 defines a `Filter` object for the `GetList()` method call, filtering by a combination of `MessageTypeList` and `SenderList` properties.


Available for download on Wrox.com

LISTING 10-37: Filtering messages retrieved from an inbox

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");

var filter:Object = { MessageTypeList:["MMS"],
                      SenderList:["Alma", "Olivene", "Jai"] };

var params:Object = { Type:"Inbox", Filter:filter };
var result:Object = messaging.GetList(params);
```

Sorting Inbox Messages

In addition to filtering messages, you can also sort them. Defining a `Sort` object as one of the parameters to `GetList()` enables you to retrieve and sort the messages from an inbox in a specified order.

www.it-ebooks.info

There are two properties to define on the `Sort` object: `Sort.Key` and `Sort.Order`. The `Sort.Key` string property gives you the option of setting the criterion you want to sort by. You can set this to `Date`, `Size`, `Sender`, `Subject`, or `MessageId`. By default, the `Key` value is set to `Date`. The `Sort.Order` string property allows you to define the ordering of the message objects returned as `Ascending` or `Descending`; by default, the `Order` property is `Ascending`.

The following code snippet retrieves messages of type MMS in reverse date order, which will ensure the latest messages in the inbox are read first (see Listing 10-38).



LISTING 10-38: Sorting messages retrieved from an inbox

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");

var sort:Object = { Key:"MMS", Order:"Descending" };

var filter:Object = { MessageTypeList:["MMS"],
                      SenderList:["Alma", "Olivene", "Jai"] };

var params:Object = { Type:"Inbox",
                      Filter:filter,
                      Sort:sort };

var result:Object = messaging.GetList(params);
```

Sending Messages

To send a message from a Flash Lite application, you need to create the message item and define its properties.

Listing 10-39 demonstrates how you can use the synchronous `Send()` call to send an "SMS" with the creation of a message object called `msg`, with the `MessageType`, `To`, and `BodyText` properties defined. Here, the value for the `To` property is set to `recipient number 1.` and should be replaced by the mobile number you want to send the message to. Once the call to the Messaging Service has been initialized, the `Send()` method can be utilized to deliver the message.



LISTING 10-39: Send SMS from within an application

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");

var msg:Object = { MessageType:"SMS",
                   To:"recipient number 1.",
                   BodyText:"Is it a girl? Or is it a boy?" };

var result:Object = messaging.Send(msg);
```

www.it-ebooks.info

Listing 10-40 shows how to send an MMS with an image referenced as an attachment. Here, you see that the `Attachment` property is specified along with its `MimeType`. The attachment must be on the device.



LISTING 10-40: Sending MMS from within an application

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");

var msg:Object = { MessageType:"MMS",
                   To:"recipient number 1.",
                   Subject:"Should I include this?",
                   Attachment:"C:\\Data\Images\04012010.jpg",
                   MimeType:"image/JPEG",
                   BodyText:"Bet they bitched all day!" };

var result:Object = messaging.Send(msg);
```

The API for Messaging allows you to send multiple attachments, and send to multiple recipients, through the `MessageParam` object. For multiple recipients, you can define the `To`, `Bcc`, and `Cc` array properties as part of the `MessageParam`. For multiple attachments, you can define the `AttachmentList` property, as described earlier, where for each `Attachment` object you specify the `FileName` and `MimeType`.

Listing 10-41 shows how to include multiple attachments in a message.



LISTING 10-41: Sending to multiple recipients

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");

var msgParams:Object = { To:"recipient number 2.",
                         Cc:"recipient number 3." };

var msg:Object = { MessageType:"MMS",
                   To:"recipent number 1.",
                   MessageParam:msgParams,
                   Subject:"Should I include this?",
                   Attachment:"C:\\Data\Images\04012010.jpg",
                   MimeType:"image/JPEG",
                   BodyText:"Bet they bitched all day!" };

var result:Object = messaging.Send(msg);
```

Monitoring an Inbox for Incoming Messages

When running standalone Flash Lite applications on Nokia devices, you'll find that new incoming messages will not pop up in the same way as when the screen is idle. Using

www.it-ebooks.info

`RegisterNotification()` you can monitor the inbox of a device for incoming messages. The method is asynchronous, and in Listing 10-42 you'll notice the `onMsg()` callback method defined is supplied as an argument. The `Type` value of the `params` object needs to be set to `NewMessage`. If the service call is successful, the application will be notified of a new message item in the `ReturnValue`.



LISTING 10-42: Using the RegisterNotification method to receive notification of incoming messages from the inbox

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");
var params:Object = { Type:"NewMessage" };
var result:Object = messaging.RegisterNotification(params, onMsg);
var info:String = "";

function onMsg(transactionID:Number, eventID:String, outParams:Object){

      if (outParams.ErrorCode == 0) {

            var msg:Object = outParams.ReturnValue;
            var sender:String = msg.Sender;
            var messageType:String = msg.MessageType;
            var messageId:String = msg.MessageId;
            var subject:String = msg.Subject;
            var time:String = msg.Time;

            info += sender + " | " + subject + " | " + messageId + newline;
            info += messageType + " | " + time + newline;

      }
}
```

To stop notifications from coming through, the `CancelNotification()` method is used. Use this method to stop the notification of new messages that you set up via a `RegisterNotification()` call (see Listing 10-43).



LISTING 10-43: Using the CancelNotification method to stop receiving notifications

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");
var params:Object = { Type:"NewMessage" };
var result:Object = messaging.CancelNotification(params);
```

Modifying the Status of a Message

The `ChangeStatus()` method enables you to modify the status property of a message item to either "Read," "Unread," "Replied," or "Forwarded," although the latter two just apply to email messages.

www.it-ebooks.info

Provide the `MessageId` and the `Status` as parameters to the `ChangeStatus` method (see Listing 10-44).



LISTING 10-44: Changing the status of a message to "Read"

```
import com.nokia.lib.Service;

var messaging:Service = new Service("Service.Messaging", "IMessaging");

var params:Object = { MessageId:"1050063",
                      Status:"Read" };

var result:Object = messaging.ChangeStatus(params);
```

Removing Messages from the Inbox

Lastly, you can use the `Delete()` method to delete a message from the inbox. You need to reference the message by the `MessageId` property, when defining parameters for the `Delete()` method. If you implement this method, I recommend that you attach a warning to it, because once a message has been deleted, you cannot retrieve it (see Listing 10-45).



LISTING 10-45: Deleting messages from the inbox

```
import com.nokia.lib.Service;
var messaging:Service = new Service("Service.Messaging", "IMessaging");
var params:Object = { MessageId:"1050063" };
var result:Object = messaging.Delete(params);
```

USING MEDIA MANAGEMENT

In this section, you'll learn how to use the Media Management Services API, and create some custom classes, which you will be able to use in your projects.

In Chapter 7 you learned how to retrieve meta data from media files loaded into the Flash Lite application. You can use the Media Management Services API to retrieve meta data information about media files that are *stored* on a device, including: music, sound, images, video, and streaming URLs.

API Features

The API has two methods, `GetList()` an asynchronous method to retrieve media file information, and `Cancel()` to cancel a pending `GetList()` call.

Retrieving Meta Data Using Filter and Sort Objects

Using the `GetList()` method, you need to supply a `Filter` object as one of the parameters with the `FileType` set. This allows you to retrieve meta data information for a particular type of media.

www.it-ebooks.info

The `Filter` object for this API can have four properties defined:

- `FileType` — A string value defining the type of media file. This can be set to either `Image`, `Sound`, `Video`, `Music`, or `StreamingURL`.
- `Key` — An optional string value representing the type of information to filter the media file information to retrieve
- `StartRange` — An optional value used alongside the `Key` property
- `EndRange` — An optional value used alongside specific `Key` properties. The end ranges specified represent either the `FileSize` or `FileDate`.

FileTypes.as

Listing 10-46 demonstrates a custom class called `FileTypes` covering the file types listed for the Media Management Services API. Each of the types available is specified as a public static variable.



LISTING 10-46: The FileTypes class

```
class com.wrox.chapter10.media.FileTypes {

      public static var IMAGE:String = "Image";
      public static var SOUND:String = "Sound";
      public static var VIDEO:String = "Video";
      public static var MUSIC:String = "Music";
      public static var STREAMING_URL:String = "StreamingURL";

      public function FileTypes(){

      }
}
```

FilterVO.as

For the `Filter` object, Listing 10-47 shows a custom value object called `FilterVO`, which can be used for filtering media.



LISTING 10-47: The FilterVO class

```
class com.wrox.chapter10.media.vo.FilterVO {

      public var FileType:String;
      public var Key:String;
      public var StartRange:String;
      public var EndRange:String;

      public function FilterVO(){

      }
}
```

www.it-ebooks.info

The `Key`, `StartRange`, and `EndRange` highlighted here are optional parameters for the `Filter` object. You'll cover them in more detail shortly.

SortVO.as

As with the other services you covered, a `Sort` object can be supplied as a parameter to the `GetList()` method alongside the `Filter` object. This determines the output for the meta data returned. The `Sort` object has two properties: an `Order` property, which can be set to either "Ascending" or "Descending," and a `Key` property, which, like the `Filter.Key` property, can also be set to a value representing the type of information to sort on.

Listing 10-48 shows a custom value object called `SortVO`, which can be used for defining the sorting parameters.



LISTING 10-48: The SortVO class

```
class com.wrox.chapter10.media.vo.SortVO {

        public static var ASCENDING:String = "Ascending";
        public static var DESCENDING:String = "Descending";

        public var Key:String;
        public var Order:String;

        public function SortVO(){

        }
}
```

Keys.as

You should realize that the `Key` property can actually be defined on both `Sort` and `Filter` objects. For the `Filter` object, you additionally have to set a value to the `StartRange` property and, depending on the `Key` defined, you may have to set a value for the `EndRange`. The `Key` value for both objects can be set to one of the following:

- `FileName` — Used to filter or sort results based on the full file name. For filtering, the name of the file is specified as the `StartRange` property.
- `FileExtension` — Used to filter or sort results based on the file extension. For filtering, the `StartRange` should be set to "." + file extension, for example, ".mp3".
- `Drive` — Used to filter or sort results based on the drive. For filtering results, the `StartRange` property should be set to the drive letter + ":", for example, "c:".
- `FileSize` — Used to filter or sort results based on the file size. For filtering, both the `StartRange` and `EndRange` values need to be set in bytes.
- `FileDate` — Used to filter or sort results based on the files creation date. For filtering, both the `StartRange` and `EndRange` values to be set in the format "YYYYMMDD:HHMMSS".

www.it-ebooks.info

- `FileNameAndPath` — Used to filter or sort results based on the full path and file name. For filtering, the `StartRange` also needs to be defined.
- `SongName` — Used to filter or sort results based on the song name, when `FileType` is set to "Music". For filtering, the `StartRange` also needs to be defined.
- `Artist` — Used to filter or sort results based on the artist name, when `FileType` is set to "Music". For filtering, the `StartRange` also needs to be defined.
- `Album` — Used to filter or sort results based on the album name, when `FileType` is set to "Music". For filtering, the `StartRange` also needs to be defined.
- `Genre` — To filter or sort results based on the genre, when `FileType` is set to "Music". For filtering, the `StartRange` also needs to be defined.
- `TrackNumber` — Used to filter or sort results based on the track number, when `FileType` is set to "Music". For filtering, the `StartRange` also needs to be defined.
- `Composer` — Used to filter or sort results based on the composer, when `FileType` is set to "Music". For filtering, the `StartRange` also needs to be defined.
- `LinkFirstURL` — Used to filter or sort results based on the URL of the streaming media file. For filtering, the `StartRange` also needs to be defined.
- `MimeType` — Used to filter or sort results based on the MIME type. For filtering, the `StartRange` also needs to be defined, for example, `image/jpg`.

The custom class `Keys` simply contains each of the values that can be assigned to both `Sort` and `Filter` objects. Each property is a static variable (see Listing 10-49).



LISTING 10-49: The Keys class

```
class com.wrox.chapter10.media.Keys {

      public static var FILE_NAME:String = "FileName";
      public static var FILE_EXTENSION:String = "FileExtension";
      public static var DRIVE:String = "Drive";
      public static var FILE_SIZE:String = "FileSize";
      public static var FILE_DATE:String = "FileDate";
      public static var MIME_TYPE:String = "MimeType";
      public static var FILE_NAME_AND_PATH:String ="FileNameAndPath";
      public static var SONG_NAME:String = "SongName";
      public static var ARTIST: String = "Artist";
      public static var ALBUM:String = "Album";
      public static var GENRE:String = "Genre";
      public static var TRACK_NUMBER:String = "TrackNumber";
      public static var COMPOSER:String = "Composer";
      public static var LINK_FIRST_URL:String = "LinkFirstURL";
      public static var MIME_TYPE:String = "MimeType";

      public function Keys() {

      }
}
```

www.it-ebooks.info

MediaManagement.as

Listing 10-50 demonstrates a class based on the Media Management Services API, in which the `GetList()` method can be called and a list of files is returned in the `getListHandler()` callback method.


Available for download on Wrox.com

LISTING 10-50: The MediaManagement class

```
import com.wrox.chapter10.media.vo.FilterVO;
import com.wrox.chapter10.media.vo.SortVO;
import mx.utils.Delegate;

class com.wrox.chapter10.MediaManagement {

      private var filter:FilterVO;
      private var sort:SortVO;
      private var params:Object;
      private var service:Service;
      private var mediaList:Array;

      public function MediaManagement(filterVO:FilterVO, sortVO:SortVO) {

            EventsDispatcher.initialize(this);
            if (filterVO) filter = filterVO;
            if (sortVO) sort = sortVO;
            init();

      }

      private function init():Void {
            service = new Service("Service.MediaManagement", "IDataSource");
            service.GetList(params, Delegate.create(this, getListHandler));
      }

      private function getListHandler(tID:Number, eID:String, result:Object){

            if (result.ErrorCode == 0) {
                  mediaList = result.ReturnValue;
            }
      }

      public function getData(propName:String):Array {

            var mediaItem:Object = null;
            var dataArr:Array = [];

            do {
                  mediaItem = mediaList.next();

                  if (mediaItem != null) {
                        dataArr.push(mediaItem[propName]);
                  } else {
                        dispatchEvent(target:this, type:"onComplete");
```

www.it-ebooks.info

```
                        break;
                    }

                } while (true);

                return dataArr;
            }
        }
```

Listing 10-51 shows how both a `Filter` and `Sort` objects can be defined to retrieve music files with the `.mp3` file extension.



Available for download on Wrox.com

LISTING 10-51: Utilizing the MediaManagement class

```
import com.wrox.chapter10.MediaManagement;
import com.wrox.chapter10.media.Keys;
import com.wrox.chapter10.media.FileTypes;
import com.wrox.chapter10.media.vo.FilterVO;
import com.wrox.chapter10.media.vo.SortVO;

var mp3FilterVO:FilterVO = new FilterVO();
mp3FilterVO.FileType = FileTypes.MUSIC;
mp3FilterVO.Key = Keys.FILE_EXTENSION;
mp3FilterVO.StartRange = ".mp3";

var mp3SortVO:SortVO = new SortVO();
mp3SortVO.Key = Keys.ARTIST;
mp3SortVO.Order = SortVO.DESCENDING;

var mediaList:MediaManagement = new MediaManagement(mp3FilterVO, mp3SortVO);
mediaList.addEventListener("onComplete", onCompleteHandler);

function onCompleteHandler(){

      mediaList.getData(Keys.ARTIST);

}
```

OTHER S60 PLATFORM SERVICES

The Sensor, Logging, and SystemInfo Service APIs also have great features that you can include as part of your Flash Lite applications.

Here's a rundown of the remaining S60 Service APIs.

ActionScript Sensor Service API

The Sensor Services API will allow you to achieve some pretty cool interactions for Flash mobile applications, allowing you to take advantage of physical sensors on devices.

www.it-ebooks.info

You can use this API to detect changes to the device's axis, and when a device is double-tapped, via its accelerometer. You can also detect when a device's orientation changes and when a device has rotated, by checking its rotational properties.

For more information on this API, visit the Forum Nokia website at `http://library.forum.nokia.com/index.jsp?topic=/Flash_Lite_Developers_Library/GUID-B77C2006-879F-4AC6-B7BF-04B25B563A29.html`

ActionScript Logging Service API

Using the Logging Service API will enable you to access a device's call logs as part of your Flash mobile applications.

You can use this API to retrieve the amount of data that was sent to and from the device over a period of time. You can also read, write to, and edit call logs.

For more on this API, visit the Forum Nokia website at `http://library.forum.nokia.com/index.jsp?topic=/Flash_Lite_Developers_Library/GUID-F003B7DD-E450-49AD-B447-C5132FE47D3C.html`

ActionScript SystemInfo Service API

Using the SystemInfo Service API will enable you to access several properties of a device, including its battery state, total space, free space, and drive name. It's all about retrieving system information.

For more on this API, visit the Forum Nokia website at `http://library.forum.nokia.com/index.jsp?topic=/Flash_Lite_Developers_Library/GUID-AEB26A58-1DE2-46CB-81EC-6DB3A477B7A3.html`

SUMMARY

In this chapter, you've covered through the S60 Platform Services APIs and gained an understanding of how you can extend Flash Lite applications for Nokia devices.

In the next chapter, you'll learn how to build an application using the Location Services API, which was covered earlier in this chapter.

Forum Nokia is supported by a great Flash Lite community and provides a comprehensive reference for the services that are covered here, including demos and tutorials. For more information and resources, visit `http://www.forum.nokia.com/`.

www.it-ebooks.info

11

Creating a Weather Client

WHAT'S IN THIS CHAPTER?

- Using Google Weather and Map APIs
- Parsing JSON using ActionScript
- Using the S60 Location Service API
- Using PureMVC to build a Weather Client

There are numerous location-based service APIs that mobile developers can utilize in their applications that are open to you as a developer for consumption.

In this chapter you'll learn how to use a weather-based Web service in your applications that utilizes the location retrieved from the device.

GOOGLE APIS

In this chapter you're going to examine a Flash Lite Weather Client, combining the S60 Symbian Location Services API covered in the last chapter, and two Google APIs:

- Google Weather API
- Google Map API

Using Google Weather API

Using the Google Weather API will allow us to retrieve the following information for use in the Flash Lite application:

- Current weather conditions for a given location
- Predicted weather forecasts for the next four days

www.it-ebooks.info

The Google Weather API is not extensively documented; nevertheless, retrieving the data you need from service is straightforward.

Here's an example HTTP request to retrieve weather information for Kuala Lumpur in Malaysia:

```
http://www.google.com/ig/api?weather=Kuala Lumpur, Malaysia&hl=en
```

In this request you can see that it takes two query parameters in the form of a location `weather=Kuala Lumpur, Malaysia` and also a language `hl=en`.

Defining Location

There are several variations of the location you can assign to `weather`, including the following:

- **City, Country** — For non-U.S.-based locations use the city name and country name combination, for example "London, England", "Kingston, Jamaica", or "Kuala Lumpur, Malaysia".
- **City, State** — For U.S.-based locations use the city name and state name combination in the format "New York, NY".
- Postal Code — Assign just the postal code for the area, for example "S9 1LP".
- **Latitude and Longitude** — Supply latitude and longitude coordinates in the format "30670000,104019996".

Defining Language

The language parameter you assign to `hl` should be provided simply as a two-letter string defined by the standard language code list ISO 639-1 (`http://www.loc.gov/standards/iso639-2/php/code_list.php`). If the parameter isn't supplied the default is English `en`.

Here is a list of known supported languages for the API:

- **`en`** — English
- **`fr`** — French
- **`de`** — German
- **`es`** — Spanish
- **`ja`** — Japanese
- **`ms`** — Malay

www.it-ebooks.info

The language parameter is a useful feature, as it gives you the option of being able to provide language settings in your application.

The XML API Reply

Listing 11-1 shows the server response for the Kuala Lumpur weather information request made on September 12, 2009.



LISTING 11-1: The server response for Kuala Lumpur

```
<?xml version="1.0"?>
<xml_api_reply version="1">
      <weather module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0">
            <forecast_information>
                  <city data="Kuala Lumpur, WP Kuala Lumpur"/>
                  <postal_code data="Kuala Lumpur, Malaysia"/>
                  <latitude_e6 data=""/>
                  <longitude_e6 data=""/>
                  <forecast_date data="2009-09-12"/>
                  <current_date_time data="2009-09-12 07:00:00 +0000"/>
                  <unit_system data="US"/>
            </forecast_information>
            <current_conditions>
                  <condition data="Mostly Cloudy"/>
                  <temp_f data="90"/>
                  <temp_c data="32"/>
                  <humidity data="Humidity: 66%"/>
                  <icon data="/ig/images/weather/mostly_cloudy.gif"/>
                  <wind_condition data="Wind: SW at 6 mph"/>
            </current_conditions>
            <forecast_conditions>
                  <day_of_week data="Sat"/>
                  <low data="73"/>
                  <high data="91"/>
                  <icon data="/ig/images/weather/thunderstorm.gif"/>
                  <condition data="Thunderstorm"/>
            </forecast_conditions>
            <forecast_conditions>
                  <day_of_week data="Sun"/>
                  <low data="75"/>
                  <high data="89"/>
                  <icon data="/ig/images/weather/thunderstorm.gif"/>
                  <condition data="Thunderstorm"/>
            </forecast_conditions>
            <forecast_conditions>
                  <day_of_week data="Mon"/>
                  <low data="75"/>
```

continues

www.it-ebooks.info

LISTING 11-1 *(continued)*

```
                <high data="89"/>
                <icon data="/ig/images/weather/chance_of_storm.gif"/>
                <condition data="Chance of Storm"/>
            </forecast_conditions>
            <forecast_conditions>
                <day_of_week data="Tue"/>
                <low data="75"/>
                <high data="89"/>
                <icon data="/ig/images/weather/chance_of_storm.gif"/>
                <condition data="Chance of Storm"/>
            </forecast_conditions>
        </weather>
</xml_api_reply>
```

Weather Conditions

You can see that the resulting `XML` is well ordered and provides details for the current weather conditions in the `current_conditions` node , several nodes named `forecast_conditions`, representing forecast weather conditions for the four subsequent days following the day of the initial call request. We'll be using all of this information directly in the application. Parsing the XML should be easy enough, and you'll go through this in depth later.

Forecast_Information

In addition to supplying the forecast's date and time the forecast information node `forecast_information` also returns location details, including city name, postal code, latitude, and longitude. However, on numerous occasions this information was not all present and not as reliable as first thought, which presents us with a slight dilemma with respect to displaying location details in our application.

Try It Out

See what happens for yourself when you make calls to the API, changing the location or language parameters in the browser.

Take a look at the `humidity` node under `current_conditions`:

```
<humidity data="Humidity: 66%"/>
```

This is the value returned when the language setting is English. When the language setting is `Japanese` the following `humidity` node is returned:

```
<humidity data="**: 74%"/>
```

www.it-ebooks.info

In Spanish the word “humidity” is replaced with “humedad”, in French you’ll see “humidité”, while in Malay you’ll see “Kelembapan.”

When using the API you, therefore, need to be aware of any changes resulting from changing the language.

Weather Icons

Each current condition and forecast conditions node returns the weather `icon` node, which has a relative path to the graphical representation of the weather condition, in the `data` attribute.

Here’s an example for “Chance of Storm”:

```
<icon data="/ig/images/weather/chance_of_storm.gif"/>
```

Each weather condition has its own corresponding icon. Here are a few examples of weather conditions returned by the API:

- Sunny
- Cloudy
- Mostly Cloudy
- Thunderstorm
- Rain
- Chance of Rain

There are several options to represent each weather condition in the application. To use the image paths retrieved from the API call, you could make a `loadMovie()` call to the absolute path of the image files. This would need to include the full Google domain, for example, “Chance of Rain.”

```
http://www.google.com/ig/images/weather/chance_of_rain.gif
```

Or, you could replicate the folder structure so that the images were relative to the main `.swf` file.

While it’s perfectly possible to load these images into the application, doing so would mean that each time a user’s request was fulfilled the application would need to make additional network requests to retrieve each of the images. With the forecast conditions that would mean four images would need to be loaded from the server.

The weather conditions listed will each be represented in the application, and there will be two versions, one for day and one for night. The graphics for the images don’t need to change. They are simply a representation of the state of weather, so it makes sense to have the weather icons on the device.

As it turns out, the Google weather icons are 40 pixels wide by 40 pixels high and too small for our application. The application will, therefore, use images relative to the application, and they can be found in the images folder accompanying this chapter.

www.it-ebooks.info

The application will use the bespoked and rather cool Tick Weather Icons designed by M.Long (`http://maxclassic.blogspot.com/`).

One of the many challenges we'll be addressing is how to replace the image paths returned from the Weather API with these icons.

Using Google Map API

Using the Google Map API (`http://code.google.com/apis/maps/`) will allow you to retrieve more specific information about a location, including the address, locality, country, post code, and geographical coordinates. This is exactly what you need for this application, so when you request weather information, you can be sure that the location is also accurate.

In our application the aim is to use the Google Maps API to:

- Perform a search for a location or place
- Supply the result of the call to the weather API
- Display the result in the application

Signing Up for a Google API Key

To use this API, you will first need to register for a Google account. Second, you need to sign up for an API Key, so that the calls made to the service can be tied to the domain using it.

The Google Map API includes a range of functions and methods to obtain detailed information about geographical location, depending on your service.

Before you move on to developing the application, you'll learn about the geocoding services provided in the API. The service retrieves accurate address information.

Geocoding

Geocoding is the process of converting "human-readable" addresses into geographic coordinates that can be interpreted and displayed on a map.

You can directly access Google's geocoder using a parameterized HTTP request (`http://maps.google.com/maps/geo?`) to the Google Maps API Geocoding Service.

Here is a geocoding example of a parameterized request for the Louvre in Paris, France:

```
http://maps.google.com/maps/geo?q=Louvre&output=xml
```

The parameters specified after `geo?` are simply the query `q` and the format `output`. The query `q` can be any location you want to search for. This can be the name of a landmark, place name, city, country, and so forth. What you specify here will determine the output as you'll see later.

Let's take a look at the format options available for the `output` parameter in more detail, as this will determine how information is processed by the application.

www.it-ebooks.info

Selecting the Response Format

The Google Map API can output responses from the server via one of the following file formats:

- Comma-separated values (CSV)
- Keyhole Markup Language (KML)
- JavaScript Object Notation (JSON)

In the CSV response for the Louvre's geocoding request, you'll see four values: geocode status, accuracy, latitude, and longitude. The geocode *status* has a value of 200, which means that the request was successful and no errors were returned. The accuracy value of 9 is the highest level of the accuracy values returned. The latitude value of 48.8608740, and longitude value of 2.3365580, detail the geo-coordinates for the location (see Listing 11-2).



LISTING 11-2: The CSV output

Available for download on Wrox.com

```
200,9,48.8608740,2.3365580
```

Of the three output options, the CSV format provides the shortest response. The KML response format is designed for geographical data and is based on XML. It has more structured information in comparison (see Listing 11-3).



LISTING 11-3: The KML output

Available for download on Wrox.com

```
<kml xmlns="http://earth.google.com/kml/2.0">
<Response>
      <name>Louvre</name>
      <Status>
            <code>200</code>
            <request>geocode</request>
      </Status>
      <Placemark id="p1">
            <address>Louvre, Paris, France</address>
            <AddressDetails Accuracy="9"
                        xmlns="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
                  <Country>
                        <CountryNameCode>FR</CountryNameCode>
                        <CountryName>France</CountryName>
                        <AdministrativeArea>
                              <AdministrativeAreaName>Ile-de-France
                                    </AdministrativeAreaName>
                              <SubAdministrativeArea>
                                    <SubAdministrativeAreaName>Paris
                                          </SubAdministrativeAreaName>
                                    <Locality>
```

continues

www.it-ebooks.info

LISTING 11-3 *(continued)*

```
                                        <LocalityName>Paris</LocalityName>
                                        <AddressLine>Louvre</AddressLine>
                                    </Locality>
                                </SubAdministrativeArea>
                            </AdministrativeArea>
                        </Country>
                    </AddressDetails>
                    <ExtendedData>
                        <LatLonBox north="48.8682142"
                                   south="48.8535327"
                                   east="2.3525654"
                                   west="2.3205506" />
                    </ExtendedData>
                    <Point>
                        <coordinates>2.3365580,48.8608740,0</coordinates>
                    </Point>
                </Placemark>
</Response>
</kml>
```

The response in JSON returns the same detail as the KML format (see Listing 11-4).

LISTING 11-4: The JSON output

Available for download on Wrox.com

```
{
    "name": "Louvre",
    "Status": {
        "code": 200,
        "request": "geocode"
    },
     "Placemark": [ {
         "id": "p1",
         "address": "Louvre, Paris, France",
         "AddressDetails": {
             "Accuracy": 9,
             "Country": {
                 "AdministrativeArea": {
                 "AdministrativeAreaName": "Île-de-France",
                 "SubAdministrativeArea": {
                     "Locality": {
                         "AddressLine": [ "Louvre" ],
                         "LocalityName": "Paris"
                     },
                     "SubAdministrativeAreaName": "Paris"
                 }
             },
             "CountryName": "France",
             "CountryNameCode": "FR"
         }
     },
```

www.it-ebooks.info

```
            "ExtendedData": {
                  "LatLonBox": {
                        "north": 48.8682142,
                        "south": 48.8535327,
                        "east": 2.3525654,
                        "west": 2.3205506
                  }
            },
            "Point": {
                  "coordinates": [ 2.3365580, 48.8608740, 0 ]
            }
      } ]
}
```

If you recall the JSON format was briefly mentioned in Chapter 9. The JSON format is used for the Weather Client, so you'll take a more detailed look at the format and how to use it next.

Parsing JSON

To parse the JSON format in AS2, you can use the `JSON` class. This can be downloaded from the JSON.org Web site (`http://www.json.org/json.as`); the original class here was ported to AS2 by Trannie Carter.

The following steps show how responses from the Google Maps API returning the JSON format are parsed. At the end you'll have a `.swf` file that allows you to return the value of a JSON object by property name. The approach will utilize the `LoadVars` class.

You'll be creating the `chapter11_JSON.fla` example, which you can also find in the `.zip` file accompanying this chapter. All the code created here is found in the `.fla` file with the exception of the JSON class. You can compile the example and run it as it is, or you can follow the walkthrough to gain a better understanding of how parsing JSON is achieved.

1. Start by creating a new Flash mobile file for the Nokia E71. This is used here simply to provide the 320 × 240 px dimensions. Save the file as `chapter11_JSON.fla`.
2. Ensure that the `JSON` class is in the same directory as your `.fla` file, and then import the class along with the `Delegate` class to the first frame of the Timeline.
3. Under the import statements, instantiate a new `JSON` object called `json`. Add four more variables: an object called `jsonObject`; a string to hold the parsed JSON result called `jsonString`; a variable for the URL, called `jsonURL`; and lastly a `LoadVars` instance named `jsonRequest`. Ensure that the geocoding query for the Louvre is assigned as the URL for `jsonURL` (see Listing 11-5).



LISTING 11-5: Creating a JSON request

```
import JSON;
import mx.utils.Delegate;

var json:JSON = new JSON();
var jsonObject:Object;
```

continues

www.it-ebooks.info

LISTING 11-5 *(continued)*

```
var jsonString:String;
var jsonURL:String = "http://maps.google.com/maps/geo?q=Louvre&output=json";

var jsonRequest:LoadVars = new LoadVars();
```

4. Next, create a handler for the `onLoad` event of `jsonRequest` called `onLoadComplete()`. The `onLoadComplete()` method is where you'll be calling the `parse()` method of `json`, which, as the name suggests, does the actual parsing of a JSON object. Following the `onLoad()` assignment, make a call to the `load()` method supplying the `jsonURL` (see Listing 11-6).



Available for download on Wrox.com

LISTING 11-6: Using the load method of LoadVars to retrieve the JSON object

```
var json:JSON = new JSON();
var jsonObject:Object;
var jsonString:String;
var jsonURL:String = "http://maps.google.com/maps/geo?q=Louvre&output=json";

var jsonRequest:LoadVars = new LoadVars();
jsonRequest.onLoad = Delegate.create(this, onLoadComplete);
jsonRequest.load(jsonURL);

function onLoadComplete(success) {
      if(success) {
            trace("jsonRequest.toString:: " + jsonRequest + "END");
            var unescapedStr:String = unescape(jsonRequest.toString());
            jsonObject = json.parse(unescape(unescapedStr));
      } else {
            trace("unable to load JSON data");
      }
}
```

Examine the lines inside `onLoadComplete()` a little closer.

The `load()` method of `jsonRequest` is what triggers the request. This call should ideally be placed on a button click; however, in this example the request is made when you run the example. The specified JSON call actually returns the following URL-encoded string:

```
%7B%0A%20%20%22name%22%3A%20%22Louvre%22%2C%0A%20%20%22Status%22%3A%20%7B%0A%2
0%20%20%20%22code%22%3A%20200%2C%0A%20%20%20%20%22request%22%3A%20%22geocode%2
2%0A%20%20%7D%2C%0A%20%20%22Placemark%22%3A%20%5B%20%7B%0A%20%20%20%20%22id%22
%3A%20%22p1%22%2C%0A%20%20%20%20%22address%22%3A%20%22Louvre%2C%20Paris%2C%20F
rance%22%2C%0A%20%20%20%20%22AddressDetails%22%3A%20%7B%0A%20%20%20%22Accuracy
%22%20%3A%209%2C%0A%20%20%20%22Country%22%20%3A%20%7B%0A%20%20%20%20%20%20%22A
dministrativeArea%22%20%3A%20%7B%0A%20%20%20%20%20%20%20%20%20%22Administrativ
eAreaName%22%20%3A%20%22%CD%ACe%2Dde%2DFrance%22%2C%0A%20%20%20%20%20%20%20%2
0%20%22SubAdministrativeArea%22%20%3A%20%7B%0A%20%20%20%20%20%20%20%20%20%20%
20%20%22Locality%22%20%3A%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
```

www.it-ebooks.info

```
20%22AddressLine%22%20%3A%20%5B%20%22Louvre%22%20%5D%2C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22LocalityName%22%20%3A%20%22Paris%22%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D%2C%0A%20%20%20%20%20%20%20%20%20%20%20%20%22SubAdministrativeAreaName%22%20%3A%20%22Paris%22%0A%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%7D%2C%0A%20%20%20%20%20%20%22CountryName%22%20%3A%20%22France%22%2C%0A%20%20%20%20%20%20%22CountryNameCode%22%20%3A%20%22FR%22%0A%20%20%20%7D%0A%7D%2C%0A%20%20%20%20%22ExtendedData%22%3A%20%7B%0A%20%20%20%20%20%20%22LatLonBox%22%3A%20%7B%0A%20%20%20%20%20%20%20%20%22north%22%3A%2048%2E8682142%2C%0A%20%20%20%20%20%20%20%20%22south%22%3A%2048%2E8535327%2C%0A%20%20%20%20%20%20%20%20%22east%22%3A%202%2E3525654%2C%0A%20%20%20%20%20%20%20%20%22west%22%3A%202%2E3205506%0A%20%20%20%20%20%20%7D%0A%20%20%20%20%7D%2C%0A%20%20%20%22Point%22%3A%20%7B%0A%20%20%20%20%20%20%22coordinates%22%3A%20%5B%202%2E3365580%2C%2048%2E8608740%2C%200%20%5D%0A%20%20%20%20%7D%0A%20%20%7D%20%5D%0A%7D%0A=&onLoad=%5Btype%20Function%5D
```

If you run the example at this stage, you will see in the output window that the result is a sequence of hexadecimal values; the majority is white space "%20". Somewhere in there is the readable string you want to use to pass onto the JSON parser.

Using the `unescape()` method you can decode the value you need from the URL-encoded format into ASCII characters so that it can be parsed correctly by the parser. In the `onLoadComplete()` method notice that the result of the `unescape()` operation is assigned to a variable called `unescapedStr`, which is then passed onto the `parse()` method of the `JSON` instance `json`. This is all done within the `success` portion of the if statement for `onLoadComplete()`. Also notice that there are two `trace()` statements used, the first to return the output of the unescaped JSON result, and the second to declare the request unsuccessful.

5. Next, add the visual elements to the stage, three text fields and a single push button. One text field needs to be an input field with the name of `property_txt`. This will be used to specify the name of the property to retrieve from the JSON object. The second and third text fields should be dynamic, one called `request_txt`, which will display the request made, and the other `value_txt`, to display the value of the property. Add the button to the stage also, and give it an instance name of `jsonBtn`. Shortly this will make a call to a function called `onSelectHandler()`.

6. Next create a function called `writeJsonValue()`, to write a JSON property to `value_txt` (see Listing 11-7).



LISTING 11-7: Writing to the json value text field

```
function writeJsonValue(value:Object) {
        value_txt.text = value;
}
```

7. Next, create the `getJsonObject()` function. This needs three parameters. The first parameter, called `jsonObj`, is an object . The second parameter, `jsonSearch`, is the `JSON` property you want to retrieve the value for from `jsonObj`, and the third property,

www.it-ebooks.info

`jsonProp`, is a reference to the property calling the method. The method is a *recursive* function because it makes a call to itself to iterate through each of the properties in the JSON string (see Listing 11-8).



LISTING 11-8: The getJsonObject method recursive function

Available for download on Wrox.com

```
function getJsonObject(jsonObj:Object, jsonSearch:String, jsonProp:String):Object {

      var nextObj:Object;
      var jsonValue:Object;

      for (var jsonPropName:String in jsonObj) {

            jsonValue = jsonObj[jsonPropName];

            if(jsonPropName == jsonSearch) {
                  writeJsonValue(jsonValue);
            }

            nextObj += getJsonObject(jsonValue, jsonSearch, jsonPropName);
      }

      return jsonValue;
}
```

Within `getJsonObject()` two variables are defined, `nextObj` and `jsonValue`. Both are objects. The `for` loop in the function uses the variable `jsonPropName` to go through each of the objects it finds in `jsonObj`, until it has gone through them all. Each property value found in `jsonObj` is saved to `jsonValue`, which is returned by the function in the last line. There is also an `if` statement in the `for` loop to handle when the property name `jsonPropName` matches the `jsonSearch` argument. At this point, the function calls the `writeJsonValue()` function, passing the `jsonValue`. Following the `if` statement you can see that the `getJsonObject()` function is called within itself, with each of the variables so far mentioned passed as arguments: `jsonValue`, `jsonSearch`, and `jsonPropName`. The result of the call, which should be the `jsonValue`, is assigned to `nextObj`.

8. Finally, create the `onSelectHandler()` and assign it to the push button's `onSelect()` event (see Listing 11-9).



LISTING 11-9: The PushButton's onSelectHandler method

Available for download on Wrox.com

```
function onSelectHandler() {

      var propName:String = property_txt.text;

      getJsonObject(jsonObject, propName, "this");
```

www.it-ebooks.info

```
        request_txt.text = "getJsonObject(jsonObject, \"" + propName + "\", " +
"\"name\"" +");";
}

jsonBtn.onSelect = Delegate.create(this, onSelectHandler);
```

In the `onSelectHandler()` function, you'll see that the text from the `property_txt` text field is assigned to the variable `propName`. The `getJsonObject()` method is then called, with the variables `jsonObject` and `propName` supplied as parameters. The `request_txt` is also assigned the call.

You should now be able to run the example and enter some of the properties returned by the JSON request. For example, entering "address" will return "Louvre, Paris, France".

The majority of the code you've just created in this section is used in the `JSONHelper` class.

WIREFRAMES AND DESIGN

Wireframes help to communicate how the application will look with minimal design. For the Weather Client there are a number of wireframes detailing the views used in the application. Figure 11-1 illustrates these screens.

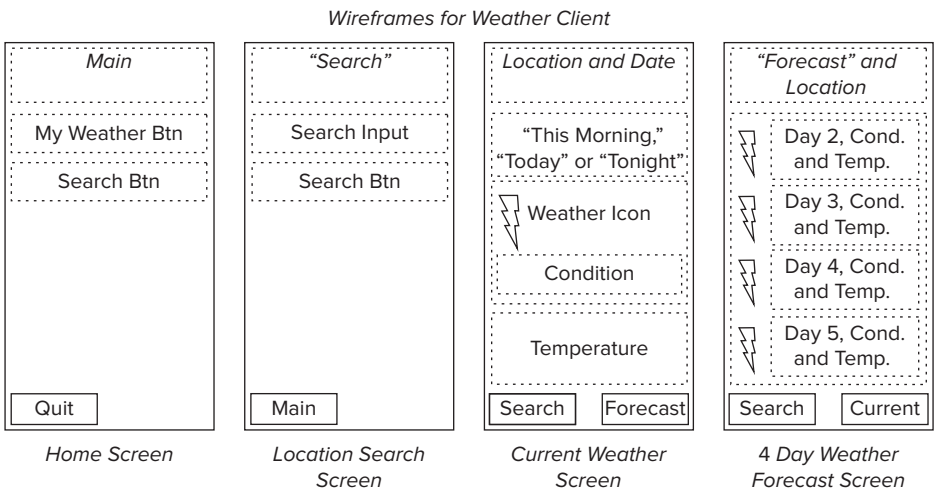


FIGURE 11-1

- **Home Screen —** Consists of My Weather and Search options
- **Search Screen —** Consists of the search field and submit button
- **Current Weather Screen —** Consists of a Location, date, the weather icon, current conditions, and the time of day
- **4 Day Weather Forecast Screen —** Consists of four days of weather information

www.it-ebooks.info

In each screen, notice that the soft keys have been labeled; these should give you an indication as to the notifications each mediator will send for navigation. Figure 11-2 shows a mockup design for the current weather conditions screen. Here you can see how the wireframe translates.

Figure 11-3 shows a mockup design for the Forecast screen.



FIGURE 11-2



FIGURE 11-3

In the next section you'll cover each aspect of the build for both the current weather view and the 4 day forecast.

BUILDING THE APPLICATION

Now you can explore the code behind the Weather Client, covering the following aspects of building the application:

- The `.fla` file
- Defining ApplicationFacade
- Creating a Model for the Weather Client

www.it-ebooks.info

- Creating the Weather Client's Controller
- Exploring the View's Mediators

Before you begin, ensure you have access to all the files associated with this chapter.

The .fla File

Open the `chapter11_WeatherClient.fla` file in Flash CS4. In the Library panel for the application, you'll see that there are several assets, all of which are used in the application.

On the stage in the `.fla` you'll find the background for the application. The MovieClip component has an instance name of `background_mc`, and is found in the "background folder" of the Library, which also contains `day`, `afternoon`, and `night` assets, forming the `background` seen on the stage.

The `temperature_txt` asset consists of the text for the temperature in the current conditions view. The Arial font is embedded into the application via this asset.



Because the majority of the components and mediators for the Weather Client embed the Arial font, you will need to ensure that it is either present on your system, or replace it with a font that is.

The `degrees_mc` asset is used for displaying the temperature in Fahrenheit and is added to the application via the `CurrentConditionsComponent`. There's also a `SoftKeys` component, added to the application by the `SoftKeysMediator`. The `PushButton` component can also be found in the Library, and this is used by the `OptionsMediator` and the `SearchMediator` — more on the mediators later. The line `separator` asset is used again also.

On the first frame of the Timeline in the `.fla` file, you'll find the call to kick start the application:




```
import com.wrox.chapter11.ApplicationFacade;
var app:ApplicationFacade = ApplicationFacade.getInstance();
app.startup(this);
```

code snippet chapter11_WeatherClient.fla

Defining ApplicationFacade

The `ApplicationFacade` class for the application can be broken into the methods `startup()`, `getInstance()`, and `initializeController()`, and notification names.

Defining the Notification Names

The `ApplicationFacade` for the Weather Client contains 15 notifications names, which are used by various aspects of the model, view, and controller. You'll learn more about how these are utilized over the remainder of this chapter (see Listing 11-10).

www.it-ebooks.info



LISTING 11-10: The notification names defined in the Weather Client's ApplicationFacade

Available for download on Wrox.com

```
public static var GET_GOOGLE_WEATHER:String = "getGoogleWeather";
public static var GET_GOOGLE_WEATHER_SUCCESS:String = "getGoogleWeatherSuccess";
public static var GET_GOOGLE_WEATHER_FAILURE:String = "getGoogleWeatherFailure";
public static var GET_HANDSET_LOCATION:String = "getHandsetLocation";
public static var GET_HANDSET_LOCATION_FAILURE:String = "getHandsetLocationFailure";
public static var GET_MAP_LOCATION:String = "getMapLocation";
public static var SHOW_SEARCH:String = "showSearch";
public static var STARTUP:String = "startup";
public static var TIME_UPDATED:String = "timeUpdated";
public static var UPDATE_SOFT_KEYS:String = "updateSoftKeys";
public static var VIEW_FORECAST:String = "viewForecast";
public static var VIEW_FORECAST_CONDITIONS:String = "viewForecastConditions";
public static var VIEW_CURRENT:String = "viewCurrent";
public static var VIEW_CURRENT_CONDITIONS:String = "viewCurrentConditions";
public static var QUIT:String = "quit";
```

The notification names should be self-explanatory. The `GET_GOOGLE_WEATHER` notification name represents the call to retrieve the weather using the Google API, whereas the `GET_HANDSET_LOCATION` notification represents the call to retrieve the handset location using the S60 Service Location API.

Notification names, such as `STARTUP`, `UPDATE_SOFT_KEYS`, and `QUIT`, were used in earlier chapters and should be familiar to you.

The `initializeController()` method registers seven of the commands with the controller when the application is initialized (see Listing 11-11).



LISTING 11-11: Registering commands in the initializeController method

Available for download on Wrox.com

```
public function initializeController():Void {
      super.initializeController();
      registerCommand(GET_GOOGLE_WEATHER, GetGoogleWeatherCommand);
      registerCommand(GET_MAP_LOCATION, GetMapLocationCommand);
      registerCommand(GET_HANDSET_LOCATION, GetHandsetLocationCommand);
      registerCommand(STARTUP, StartupCommand);
      registerCommand(VIEW_CURRENT_CONDITIONS, ViewCurrentConditionsCommand);
      registerCommand(VIEW_FORECAST_CONDITIONS, ViewForecastConditionsCommand);
      registerCommand(QUIT, QuitCommand);
}
```

You'll cover each of these commands later in this chapter. Next up is creating the model.

Creating a Model for the Weather Client

In this section you'll examine the model defined for the application (see Figure 11-4).

www.it-ebooks.info

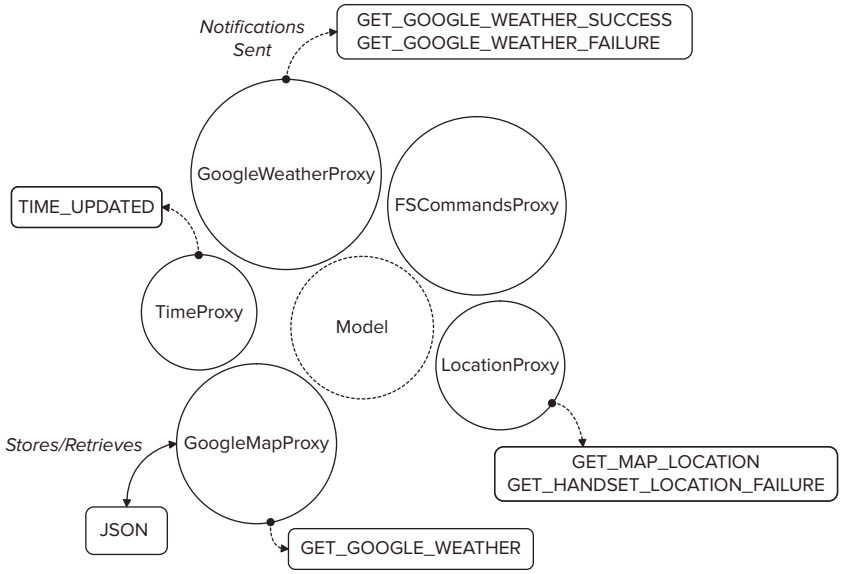


FIGURE 11-4

In Figure 11-4, you can see four new proxy classes that are used for the Weather Client application, and each of these has its own responsibility:

- `GoogleWeatherProxy` — Provides methods for retrieving weather conditions for a location from Google
- `GoogleMapProxy` — Provides methods for retrieving a location, based on a search query to Google
- `LocationProxy` — Provides the location of the handset, which is retrieved via the Nokia S60 Service API
- `TimeProxy` — Provides the time set on the device and sets the time of day

In total, there are five proxy classes used in the application. There is also the `FSCommandsProxy`, which has one additional change to the version that was discussed in previous chapters. This will be covered later.

The Project panel, shown in Figure 11-5, indicates the file system layout for the Weather Client, highlighting each of the files in the model directory.

Before looking at the new proxy classes in detail, take a look at the two value objects used by the model: `ConditionVO` and `WeatherVO`.



FIGURE 11-5

www.it-ebooks.info

ConditionVO

The `ConditionVO` class contains 12 public variables, including a variable for the weather condition, called `condition`; a variable for the temperature in Celsius, called `temp_c`; and a variable for Fahrenheit called, `temp_f`. For clarity, these variables have the same naming convention as their respective node names as seen in the XML covered earlier (see Listing 11-12). There are additional properties in the `ConditionVO` class, which have been added for use as part of the data model, including a variable for the time of day called `time_of_day`, a variable to hold reference to a small weather icon called `icon_sml`, and also a variable to hold reference to a large weather icon called `icon_lrg`.



LISTING 11-12: The ConditionVO class

Available for download on Wrox.com

```
class com.wrox.chapter11.model.data.vo.ConditionVO {

        public var condition:String;
        public var temp_f:String;
        public var temp_c:String;
        public var humidity:String;
        public var icon:String;
        public var wind_condition:String;
        public var day_of_week:String;
        public var low:String;
        public var high:String;
        public var time_of_day:String;
        public var icon_lrg:String;
        public var icon_sml:String;

        public function ConditionVO() {

        }
}
```

WeatherVO

In the `WeatherVO` class, you will notice that there are four properties: an address variable named `address`, a variable to reference the date called `date`, a variable to reference the current weather conditions called `currentConditions`, and lastly the forecast conditions variable called `forecastConditions`. The `forecastConditions` is required to hold an array of `ConditionVO` value objects (see Listing 11-13).



LISTING 11-13: Properties of the WeatherVO class

Available for download on Wrox.com

```
import com.wrox.chapter11.model.data.vo.ConditionVO;

class com.wrox.chapter11.model.data.vo.WeatherVO {

        public var address:String;
        public var currentConditions:ConditionVO;
```

www.it-ebooks.info

```
        public var date:String;
        public var forecastConditions:Array;

        public function WeatherVO() {
              currentConditions = new ConditionVO();
              forecastConditions = [];
        }
  }
```

Each value of `WeatherVO` will be parsed through `GoogleWeatherProxy`. You'll take a look at this next.

Retrieving the Weather

The `GoogleWeatherProxy` is responsible for providing the application with weather data. It will be used for retrieving the current weather, forecast conditions, and temperatures. The model for the weather data will use `WeatherVO`, `ConditionVO` and `ForecastConditionsVO`, which you'll cover shortly.

The `GoogleWeatherProxy` class utilizes an instance of the `GoogleWeatherLoader` class called `loader`, to load data from the Weather API, and also an instance of the `GoogleWeatherParser` class called `parser`, to parse the XML result of the initial load.

GoogleWeatherLoader.as

The `GoogleWeatherLoader` class uses `EventDispatcher`, allowing it to dispatch events. To ensure that this is activated, the static `EventDispatcher.initialize()` method is called in the constructor, where a reference to the `GoogleWeatherLoader` is supplied as the target for initialization. This implementation should look familiar (see Listing 11-14).



LISTING 11-14: The constructor for GoogleWeatherLoader

```
public function GoogleWeatherLoader() {
      EventDispatcher.initialize(this);
}
```

The class has a private variable called `xml`, and data is loaded via the `xml.load()` method to the XML format when the public `init()` method is called on an instance of the class. This simply requires the `url` string, which needs to be the full path to the Weather API. The public `init()` method is where the request to load weather data is initialized (see Listing 11-15).



LISTING 11-15: The init method for GoogleWeatherLoader

```
public function init(url:String):Void {
      xml = new XML();
      xml.onLoad = Delegate.create(this, onRequestComplete);
      xml.ignoreWhite = true;
      xml.load(url);
}
```

www.it-ebooks.info

You should notice here that the `onLoad` event is handled by `onRequestComplete()`. In this method the `onLoadCompleted` and `onFailed` events are dispatched. The `success` parameter indicates whether the request has been successful or not (see Listing 11-16).



LISTING 11-16: The onRequestComplete method for GoogleWeatherLoader

Available for download on Wrox.com

```
private function onRequestComplete(success:Boolean):Void {
      if (success) {
            dispatchEvent({target:this, type:"onLoadCompleted"});
      } else {
            dispatchEvent({target:this, type:"onFailed"});
      }
}
```

The `getData()` method of `GoogleWeatherLoader` can be called in `GoogleWeatherProxy` after it receives the `onLoadCompleted` event. This essentially contains the XML (see Listing 11-17).



LISTING 11-17: The getData method for GoogleWeatherLoader

Available for download on Wrox.com

```
public function getData():XML {
      return xml;
}
```

You've now covered each aspect of `GoogleWeatherLoader`. The data loaded by the class is retrieved via the public `getData()` method is passed onto the parser.

GoogleWeatherParser.as

The `GoogleWeatherParser` class also uses the `EventDispatcher` class so that it can dispatch events. Here the `onParseCompleted` event is fired when parsing of the XML is completed.

The `GoogleWeatherParser` has several methods to help with parsing data from the XML into the value objects `ConditionVO` and `WeatherVO`.

There are two key aspects of the `GoogleWeatherParser` that you should be aware of. The first relates to the use of images in the application. The weather client does not use the images returned from the API. This allows you to use your own resources for the weather data that is parsed. The second aspect relates to the days of the week. The weather client will display the full name of the day in the application rather than an abbreviation, which is parsed. Currently, the API returns the forecast day values as three-letter abbreviations. So `GoogleWeatherParser` will covert "Sun" to "Sunday".

There are two methods in `GoogleWeatherParser` to account for these points:

- `imagePathHelper()` — Replaces the image path returned from the Google Weather API request and updates it to a local reference on the `ConditionVO`
- `dayHelper()` — Replaces the three-letter day abbreviation on the `ConditionVO` value

www.it-ebooks.info

The `imagePathHelper()` method assists the parser to set the small and large image paths on each instance of the `ConditionVO` objects generated by the `parseCurrentConditions()` method. The method takes a `ConditionVO` as a parameter with the variable name `cVO`. Consider the example for the "sunny" weather condition. Essentially, the `imagePathHelper()` will alter the image path string from "/ig/images/weather/sunny.gif" to "images/n/sunny_sml.png". However, because there is a design requirement for two icons, one for the small image and one for the large, the actual file names that need to be created are "images/n/sunny_sml.png" and "images/n/sunny_lrg.png", note that these are paths relative to the location of the main `.swf` file (see Listing 11-18).



LISTING 11-18: The imagePathHelper method for GoogleWeatherParser

```
private function imagePathHelper(cVO:ConditionVO):ConditionVO {

        var oldPath:String = "/ig/images/weather/";
        var newPath:String = "images/" + dayOrNight + "/";
        var imagePath:String = cVO.icon.split(oldPath).join(newPath);

        cVO.icon_sml = imagePath.split(".gif").join("_sml.png");
        cVO.icon_lrg = imagePath.split(".gif").join("_lrg.png");

        return cVO;
    }
```

In Listing 11-18 you'll see one way to replace part of a string value combining the use of `split()` and `join()` methods. Another bonus to having this method called here is that the device images local to the application can be assigned the "day" folder path: "images/**d**/" or the "night" path: "images/**n**/".

How does this work? Basically the `imagePathHelper()` method is called after the data has been set to the `conditionVO.icon` property, so if this value were "/ig/images/weather/cloudy.gif", the `split()` method would divide the string into an array, where "/ig/images/weather/" is the point of separation. The first array value would be a blank string "", because there is nothing before the separation point. The second value in the array would then contain just the file name "cloudy.gif". The `join()` method then concatenates the "images/n/" portion, presuming the `dayOrNight` value is "n", with the file name, resulting in "images/n/cloudy.gif". This is then saved to the `imagePath` variable.

Before the `imagePathHelper()` method returns the `cVO` value object, it assigns the image path values for `cVO.icon_sml` and `cVO.icon_lrg`. The images used are in ".png" format, so the image path ".gif" needs to be replaced with ".png".

The `dayHelper()` method is less complex and simply uses a statement to check for which day of the week has been returned in the forecast conditions `day_of_week` attribute via the `parseForecastConditions()` method (see Listing 11-19).

www.it-ebooks.info



LISTING 11-19: The dayHelper method for GoogleWeatherParser

Available for download on Wrox.com

```
private function dayHelper(day:String):String {
      var dayStr:String;

      switch(day) {
            case "Sun":
                  dayStr = "Sunday";
            break;
            case "Mon":
                  dayStr = "Monday";
            break;
            case "Tue":
                  dayStr = "Tuesday";
            break;
            case "Wed":
                  dayStr = "Wednesday";
            break;
            case "Thu":
                  dayStr = "Thursday";
            break;
            case "Fri":
                  dayStr = "Friday";
            break;
            case "Sat":
                  dayStr = "Saturday";
            break;
      }

      return dayStr;
}
```

The `parseXML()` method of `GoogleWeatherParser` is called from `GoogleWeatherProxy` when data has been loaded. In this method the `xml` object is parsed into a `WeatherVO` instance through the `parseWeather()` method (see Listing 11-20).



LISTING 11-20: The parseXML method for GoogleWeatherParser

Available for download on Wrox.com

```
public function parseXML(xml:XML):Void {

      var xNode:XMLNode = xml.firstChild;
      var wNode:XMLNode = XPathAPI.selectSingleNode(xNode, "xml_api_reply/weather");

      weatherVO = parseWeather(wNode);

      dispatchEvent({target:this, type:"onParseCompleted"});
}
```

In `parseWeather()` the `parseCurrentConditions()` method is used to extract the data from the "weather/current_conditions" XML node in the `ccNode` variable. Here, `imagePathHelper()` is used to ensure that the correct image paths are written (see Listing 11-21).

www.it-ebooks.info



LISTING 11-21: The parseWeather method for GoogleWeatherParser

Available for download on Wrox.com

```
private function parseWeather(node:XMLNode):WeatherVO {

      var fiNode:XMLNode = XPathAPI.selectSingleNode(node,
"weather/forecast_information");
      var ccNode:XMLNode = XPathAPI.selectSingleNode(node,
"weather/current_conditions");
      var fcNodeList:Array = XPathAPI.selectNodeList(node,
"weather/forecast_conditions");

      var wVO:WeatherVO = new WeatherVO();

      wVO.currentConditions = imagePathHelper(parseCurrentConditions(ccNode));

      for(var i:Number = 0; i < fcNodeList.length; i++){
            var cVO:ConditionVO = parseForecastConditions(fcNodeList[i]);
            wVO.forecastConditions.push(imagePathHelper(cVO));
      }

      return wVO;
}
```

The `parseCurrentConditions()` method parses all the values from the `XML` into the `ConditionVO` value object instance `cVO` and then returns this value to the `parseWeather()` method's `wVO.currentCondtions` value. There are also two *if* statements in `parseCurrentConditions()` to handle the situation in which the `icon` or `condition` nodes are returned empty (see Listing 11-22).



LISTING 11-22: The parseCurrentConditions method for GoogleWeatherParser

Available for download on Wrox.com

```
private function parseCurrentConditions(node:XMLNode):ConditionVO {

      var cVO:ConditionVO = new ConditionVO();
      var cNode:XMLNode;

      cNode = XPathAPI.selectSingleNode(node, "current_conditions/condition");
      cVO.condition = cNode.attributes.data;

      if (cVO.condition == ""){
            cVO.condition = "Dunno?";
      }

      cNode = XPathAPI.selectSingleNode(node, "current_conditions/temp_f");
      cVO.temp_f = cNode.attributes.data;
      cNode = XPathAPI.selectSingleNode(node, "current_conditions/temp_c");
      cVO.temp_c = cNode.attributes.data;
      cNode = XPathAPI.selectSingleNode(node, "current_conditions/humidity");
      cVO.humidity = cNode.attributes.data;
      cNode = XPathAPI.selectSingleNode(node, "current_conditions/icon");
      cVO.icon = cNode.attributes.data;
```

continues

www.it-ebooks.info

LISTING 11-22 *(continued)*

```
        if (cVO.icon == ""){
              cVO.icon = "images/dunno.gif";
        }

        cNode = XPathAPI.selectSingleNode(node, "current_conditions/wind_condition");
        cVO.wind_condition = cNode.attributes.data;

        return cVO;
    }
```

Notice in `parseWeather()` that there is also a conditional loop that saves the weather forecast conditions to the `forecastConditions` property of the `WeatherVO` instance `wVO`. As previously mentioned, this property contains an array of `ConditionVO` value objects. The `parseWeather()` method calls `parseForecastConditions()` for each iteration and also runs the `imagePathHelper()` to ascertain the correct image paths for the value objects (see Listing 11-23). Here, you should notice that the `dayHelper()` method is utilized in `parseForecastConditions()`.



LISTING 11-23: The parseForecastConditions method for GoogleWeatherParser

Available for download on Wrox.com

```
private function parseForecastConditions(node:XMLNode):ConditionVO {
        var fcVO:ConditionVO = new ConditionVO();
        var fcNode:XMLNode;

        fcNode= XPathAPI.selectSingleNode(node, "forecast_conditions/day_of_week");
        fcVO.day_of_week = dayHelper(fcNode.attributes.data);
        fcNode= XPathAPI.selectSingleNode(node, "forecast_conditions/low");
        fcVO.low = fcNode.attributes.data;
        fcNode= XPathAPI.selectSingleNode(node, "forecast_conditions/high");
        fcVO.high = fcNode.attributes.data;
        fcNode= XPathAPI.selectSingleNode(node, "forecast_conditions/icon");
        fcVO.icon = fcNode.attributes.data;
        fcNode= XPathAPI.selectSingleNode(node, "forecast_conditions/condition");
        fcVO.condition = fcNode.attributes.data;

        return fcVO;
    }
```

There's one more important function of `GoogleWeatherParser` that you need to be aware of and that's the `getParsedData()` method. This method returns a `WeatherVO` instance, which should be retrieved when parsing of the XML data is complete (see Listing 11-24).



LISTING 11-24: The getParsedData method for GoogleWeatherParser

Available for download on Wrox.com

```
public function getParsedData():WeatherVO {
        return weatherVO;
    }
```

www.it-ebooks.info

GoogleWeatherProxy.as

Now that you have covered `GoogleWeatherLoader` and `GoogleWeatherParser`. Returning to the `GoogleWeatherProxy`, you can now piece together and see where the loader and parsers will be used. Here, you can follow the steps to creating the `GoogleWeatherProxy` and understand the logic behind the code.

1. Begin by creating the familiar proxy structure. For `GoogleWeatherProxy` import the required classes, `GoogleWeatherLoader` and `GoogleWeatherParser`, and the value objects, `ConditionVO` and `WeatherVO`. Nine private variables must be declared: the `address`; a reference to the specific address of the location for the weather; `loader`, which is an instance of `GoogleWeatherLoader`; the `location` string; `parser`, which is an instance of `GoogleWeatherParser`; `timeOfDay`, a reference to the time of day; `dateStr`, a reference to the date; `dayOrNight`, a reference to day or night; `url`, to hold the path to the API; and `weatherVO`, an instance of the value object `WeatherVO` (see Listing 11-25).



Available for download on Wrox.com

LISTING 11-25: Creating the GoogleWeatherProxy class

```
import com.wrox.chapter11.ApplicationFacade;
import com.wrox.chapter11.model.loaders.GoogleWeatherLoader;
import com.wrox.chapter11.model.parsers.GoogleWeatherParser;
import com.wrox.chapter11.model.vo.ConditionVO;
import com.wrox.chapter11.model.vo.WeatherVO;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.GoogleWeatherProxy extends Proxy implements IProxy {

      public static var NAME:String = "GoogleWeatherProxy";

      private var address:String;
      private var dateStr:String;
      private var dayOrNight:String;
      private var loader:GoogleWeatherLoader;
      private var location:String;
      private var parser:GoogleWeatherParser;
      private var timeOfDay:String;
      private var url:String;
      private var weatherVO:WeatherVO;

      public function GoogleWeatherProxy(data:Object) {
            super(NAME, data);
      }
}
```

2. In the `onRegister()` method for `GoogleWeatherProxy`, instantiate an instance of the `GoogleWeatherLoader` and use the `addEventListener()` method to listen for the `onLoadCompleted` and `onFailed` events (see Listing 11-26).

www.it-ebooks.info



LISTING 11-26: The onRegister method for GoogleWeatherProxy

Available for download on Wrox.com

```
public function onRegister():Void {
      loader = new GoogleWeatherLoader();
      loader.addEventListener("onLoadCompleted", Delegate.create(this,
onLoadCompleted));
      loader.addEventListener("onFailed", Delegate.create(this, onLoadFailed));
}
```

3. Next create a `load()` method for the proxy to initialize `loader`. This should have five parameters: `newLocation` to define the location to search for, `newAddress` to define the address of the search, `newTimeOfDay` to set the time of day, `newDayOrNight` to set the day or night setting, and `newDateStr` to set the date string. The `location` value is cleared for each `load()` request, so that the last call isn't passed onto the new request. The `init()` method should be called, supplying the `url` and `location` variables as parameters (see Listing 11-27).



LISTING 11-27: The load method for GoogleWeatherProxy

Available for download on Wrox.com

```
public function load(newLocation:String, newAddress:String, newTimeOfDay:String,
newDayOrNight:String, newDateStr):Void {
      url = "http://www.google.com/ig/api?weather=";
      location = "";

      if(newLocation) location = newLocation;
      if(newAddress) address = newAddress;
      if(newTimeOfDay) timeOfDay = newTimeOfDay;
      if(newDayOrNight) dayOrNight = newDayOrNight;
      if(newDateStr) dateStr = newDateStr;

      loader.init(url + location);
}
```

4. Next, create the `onLoadComplete()` event handler to instantiate the parser using `loader.getData()` to pass the retrieve the raw XML and to the parser using `parser.parseXML()`. Here, you also need to supply `dayOrNight` as value setting for the parser to determine if night or day should be used for the image folder (see Listing 11-28).



LISTING 11-28: The onLoadComplete method for GoogleWeatherProxy

Available for download on Wrox.com

```
private function onLoadComplete():Void {
      parser = new GoogleWeatherParser();
      parser.addEventListener("onParseCompleted", Delegate.create(this,
onParseComplete ));
      parser.settings(dayOrNight);
      parser.parseXML(loader.getData());
}
```

www.it-ebooks.info

5. Next, you need to create the `onParseComplete()` event handler for the parser. Here the `GET_GOOGLE_WEATHER_SUCCESS` notification needs to be sent with an instance of `WeatherVO` in the notification body. This should only be sent as long as there is a value set on the `condition` property of the `ConditionVO` instance. If the `condition` property isn't defined, then the `GET_GOOGLE_WEATHER_FAILURE` notification is sent (see Listing 11-29).



LISTING 11-29: The onParseComplete method for GoogleWeatherProxy

```
private function onParseComplete():Void {
      setData(parser.getParsedData());

      var cVO:ConditionVO = ConditionVO(WeatherVO(getData()).currentConditions);
      cVO.time_of_day = timeOfDay;

      var wVO:WeatherVO = WeatherVO(getData());
      wVO.address = address;
      wVO.date = dateStr;

      if(cVO.condition){
            sendNotification(ApplicationFacade.GET_GOOGLE_WEATHER_SUCCESS, wVO);
      } else {
            sendNotification(ApplicationFacade.GET_GOOGLE_WEATHER_FAILURE);
      }
}
```

6. Next, create the `onLoadFailed()` handler method; this will be used to handle the `onFailed` event from the `GoogleWeatherLoader`. Here, you simply send the `GET_GOOGLE_WEATHER_FAILURE` notification (see Listing 11-30).



LISTING 11-30: The onLoadFailed method for GoogleWeatherProxy

```
private function onLoadFailed():Void {
      sendNotification(ApplicationFacade.GET_GOOGLE_WEATHER_FAILURE);
}
```

This completes the `GoogleWeatherProxy`. The last proxy created for the Weather Client is the `LocationProxy`, which you'll take a look at next.

Retrieving the Handset Location

In this section you'll just examine the code found in the `LocationProxy`, which attempts to retrieve handset location.

LocationProxy.as

The `LocationProxy` is actually the first proxy to be initialized in the application. The proxy makes an attempt to retrieve the user's GPS location using the S60 Location Service API. The aim of the `LocationProxy` is to notify the application of the device's coordinates and then pass these onto the `GoogleWeatherProxy` to retrieve the weather for the location.

www.it-ebooks.info

Figure 11-6 shows the overall flow and relationship between the `LocationProxy`, `GoogleMapProxy`, and `GoogleWeatherProxy`.

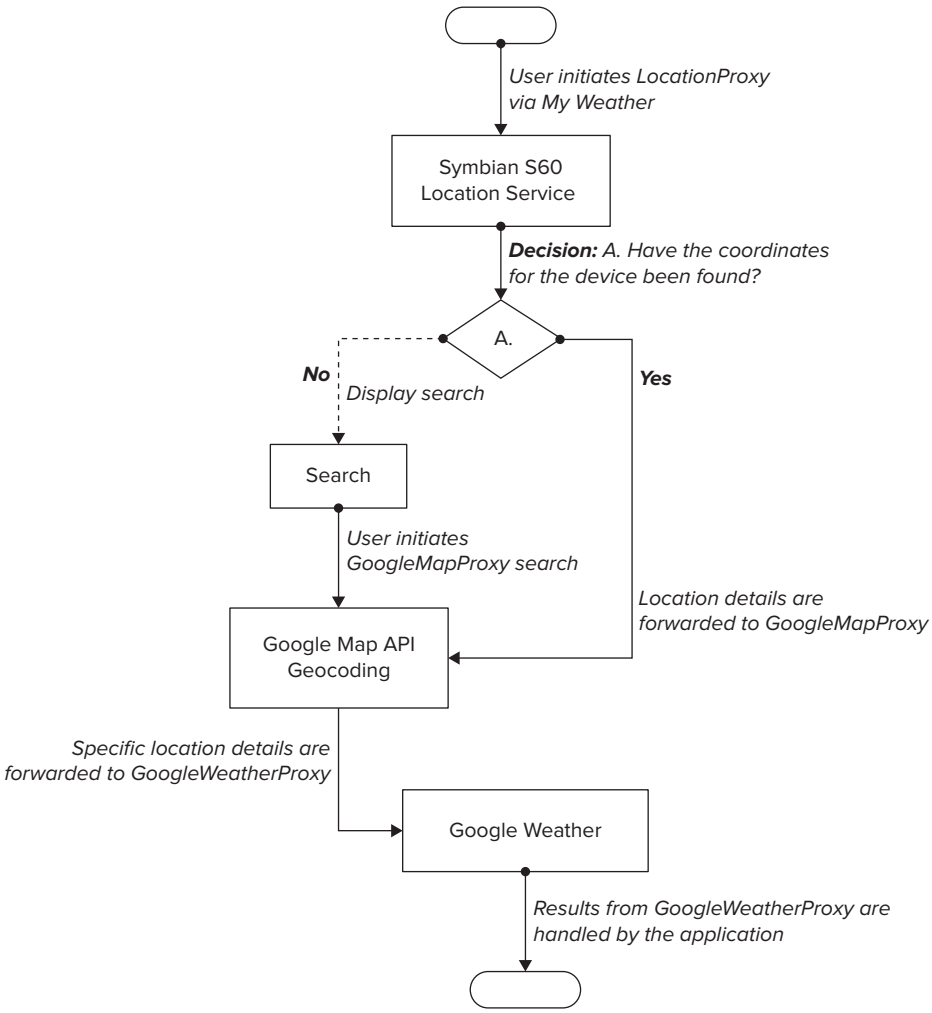


FIGURE 11-6

The `LocationProxy` has three private variables: `location`, `latitude`, and `longitude`. The `location` is an instance of the `Service` class, and `latitude` and `longitude` are strings that represent the coordinates. Notice that the `Service` class is also imported (see Listing 11-31).

www.it-ebooks.info



LISTING 11-31: Defining private variables for LocationProxy

```
import com.nokia.lib.Service;
import com.wrox.chapter11.ApplicationFacade;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.LocationProxy extends Proxy implements IProxy {

      public static var NAME:String = "LocationProxy";

      private var location:Service;
      private var latitude:String;
      private var longitude:String;

      public function LocationProxy(data:Object){
            super(NAME, data);
      }
}
```

The `LocationProxy` has a public method called `initialize()`, which is where the call to the S60 Location Service API is invoked.

To do this, the synchronous `GetLocation()` method call is made on the `location` instance, supplying just the parameter representing the type of information to retrieve from the service. The result is saved to the variable `result`. As the application is only interested in retrieving the coordinates for the location at this point, the `params` object defines the `LocationInformationClass` property as `BasicLocationInformation`. If the call doesn't return an error and `ErrorCode` is 0, the `coordinates` are retrieved and sent in the notification body of `GET_MAP_LOCATION` (see Listing 11-32).



LISTING 11-32: The initialize method for LocationProxy

```
public function initialize():Void {
      var params:Object = {LocationInformationClass:"BasicLocationInformation"};

      location = new Service("Service.Location", "ILocation");
      result = location.GetLocation(params);

      if(result.ErrorCode == 0) {

            longitude = result.ReturnValue.Longitude;
            latitude = result.ReturnValue.Latitude;

            var coordinates:String = latitude + "," + longitude;
            setData(coordinates);
```

continues

www.it-ebooks.info

LISTING 11-32 *(continued)*

```
            sendNotification(ApplicationFacade.GET_MAP_LOCATION, coordinates);

        } else {

            sendNotification(ApplicationFacade.GET_HANDSET_LOCATION_FAILURE);
        }
    }
```

If the service call results in an error and `ErrorCode` is not equal to 0, the `GET_HANDSET_LOCATION_FAILURE` notification is sent.

Because the `GetLocation()` call relies on your having a compatible device, it is more likely that the call will result in error, simply because the Location Service will be unavailable. To test the functionality in Device Central, you can alter the `initialize()` method, so that it forces the application to send the `GET_MAP_LOCATION` notification with coordinates. Listing 11-33 shows an example alteration with the coordinates supplied for the Louvre in Paris.



LISTING 11-33: Altering the initialize method in LocationProxy for testing in Device Central

Available for download on Wrox.com

```
public function initialize():Void {
        var params:Object = {LocationInformationClass:"BasicLocationInformation"};

        location = new Service("Service.Location", "ILocation");
        result = location.GetLocation(params);

        if(result.ErrorCode != 0) {

            longitude = "2.3365580";
            latitude = "48.8608740";

            var coordinates:String = latitude + "," + longitude;
            setData(coordinates);

            sendNotification(ApplicationFacade.GET_MAP_LOCATION, coordinates);

        } else {

            sendNotification(ApplicationFacade.GET_HANDSET_LOCATION_FAILURE);
        }
    }
```

Creating GoogleMapProxy.as

The `GoogleMapProxy` class is responsible for returning information about a location. It utilizes the `JSONHelper` class to retrieve information from the Google Maps API in JSON format. This aspect of the class was covered earlier in the chapter.

In this section you'll walk through the code for creating this proxy.

www.it-ebooks.info

1. First create the skeleton for the `GoogleMapProxy` class (see Listing 11-34).



Available for download on Wrox.com

LISTING 11-34: Creating the GoogleMapProxy class

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.GoogleMapProxy extends Proxy implements IProxy {

      public static var NAME:String = "GoogleMapProxy";

      public function GoogleMapProxy(data:Object) {
            super(NAME, data);
      }

      public function onRegister():Void {

      }
}
```

2. Next import the `JSONHelper` class and declare the private variables used for the proxy. You will need an instance of `JSONHelper` called `jsonObj` and four other string variables relating to the proxy requests: `requestURL`, `requestOutput`, `requestKey`, and `requestLocation`. Ensure that `jsonObj` is instantiated in the `onRegister()` method (see Listing 11-35).



Available for download on Wrox.com

LISTING 11-35: Declaring private variables in GoogleMapProxy

```
import com.wrox.chapter11.model.parsers.JSONHelper;

import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.GoogleMapProxy extends Proxy implements IProxy {

      public static var NAME:String = "GoogleMapProxy";

      private var jsonObj:JSONHelper;
      private var requestURL:String;
      private var requestOutput:String;
      private var requestKey:String;
      private var requestLocation:String;

      public function GoogleMapProxy(data:Object) {
            super(NAME, data);
```

continues

www.it-ebooks.info

LISTING 11-35 *(continued)*

```
        }

        public function onRegister():Void {
              json = new JSONHelper();
        }

}
```

We'll return to the `jsonObj` shortly. First, let's run through the private variables you just created. The `requestURL` variable will essentially be a reference to the full Google Map API call. The `requestKey` variable should contain `&key=` portion of the URL call followed by your unique Google Developer API Key. We've omitted the key here. As previously mentioned, you should use your own key value in the request call. The `requestLocation` variable will hold a reference to query location and, finally, the `requestOutput` holds a reference to the output type. Each variable is assigned a value in the `geoLocation()` method.

3. Next, create the `getLocation()` method. The method concatenates all the request parameters and then calls the `load()` method of `jsonObj` supplying the `requestURL` variable (see Listing 11-36).



Available for download on Wrox.com

LISTING 11-36: The getLocation method for GoogleMapProxy

```
public function getLocation( query:String ):Void {

        requestOutput = "&output=json";
        requestKey = "&key=";

        requestURL = "http://maps.google.com/maps/geo?q=" + query;
        requestURL += requestKey;
        requestURL += requestOutput;

        jsonObj.load(requestURL);

}
```

This method will be called from `GetMapLocationCommand`. Each time the notification name mapped to `GetMapLocationCommand` is triggered, it will call the `getLocation()` method of `GoogleMapProxy` — more on this later.

4. Ensure that you `import` the `ApplicationFacade` class and the `Delegate` class into `GoogleMapProxy`.

5. Next, create a private handler called `onParseComplete()` for the `onParseComplete` event, which is triggered from `jsonObj`. The method needs to retrieve coordinates and address data from the `jsonObj` object that made the call to the Google Maps API in step 3 (see Listing 11-37).

www.it-ebooks.info


Available for download on Wrox.com

LISTING 11-37: The onParseComplete method for GoogleMapProxy

```
private function onParseComplete(target:Object):Void {

        jsonObj.getJSONObject(target, "coordinates", "0");
        var coordinates:Array = String(jsonObj.getData()).split(",");

        jsonObj.getJSONObject(target, "address", "1");
        var address:String = String(jsonObj.getData());

        setData( {Coordinates:coordinates, Address:address} );
        sendNotification(ApplicationFacade.GET_GOOGLE_WEATHER, getData());

}
```

In the `onParseComplete()` method, you'll see that the call to `getJSONObject()` is made twice. The first call is to retrieve the coordinates for the location query. The `jsonObj` value is saved to the `coordinates` array primarily using `jsonObj.getData()`. The string returned from `getData()` is then `split(",")` into latitude and longitude values, where `coordinates[0]` is latitude and `coordinates[1]` is longitude. The second time the `getJSONObject()` method is called is to retrieve the address of the location query. The `onParseComplete` handler also saves a reference of the `jsonObj` data here as an object via the `setData()` method. The last line of the `onParseComplete` method sends the data saved by the proxy, using the `GET_GOOGLE_WEATHER` notification name.

6. Finally, in the `onRegister()` method, use the `addEventListener()` method of `jsonObj` to add a handler for the `onParseComplete` event. You will need to import the `Delegate` class at this step also (see Listing 11-38).


Available for download on Wrox.com

LISTING 11-38: The onRegister method for GoogleMapProxy

```
public function onRegister():Void {
        jsonObj = new JSONHelper();
        jsonObj.addEventListener("onParseComplete", Delegate.create(this,
onParseComplete));
}
```

Utilizing the Device Clock

When successful data requests have been made the time of day needs to be represented by the app. For this we'll create the TimeProxy.

TimeProxy.as

The `TimeProxy` class retrieves the current date from the device and determines the time of day as either morning, day, or night. You can follow the next few steps to understanding how this is achieved.

1. First, create the familiar proxy class for the `TimeProxy`, and include the `onRegister()` method (see Listing 11-39).

www.it-ebooks.info



LISTING 11-39: Creating the TimeProxy class

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.TimeProxy extends Proxy implements IProxy {

      public static var NAME:String = "TimeProxy";

      public function TimeProxy(data:Object){
            super(NAME, data);
      }

      public function onRegister() {

      }

}
```

2. Include the private variables `date`, `dayOrNight`, and `timeOfDay` (see Listing 11-40).



LISTING 11-40: Declaring private variables for the TimeProxy

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.TimeProxy extends Proxy implements IProxy {

      public static var NAME:String = "TimeProxy";

      private var date:Date;
      private var dayOrNight:String;
      private var timeOfDay:String;

      public function TimeProxy(data:Object) {
            super(NAME, data);
      }
}
```

3. Declare five static variables. The variables named `THIS_MORNING`, `TODAY`, and `TONIGHT` will be assigned to the `timeOfDay` variable and used as human-readable text for the application, depending on the time of day. Then, the `DAY` and `NIGHT` variables, which will be assigned to the `dayOrNight` variable, will be used to define which folder the images should be retrieved from. In this case, values will be either "n" for `NIGHT`, or "d" for `DAY` (see Listing 11-41).

www.it-ebooks.info



Available for download on Wrox.com

LISTING 11-41: Static variables for TimeProxy

```
public static var NAME:String = "TimeProxy";

public static var DAY:String = "d";
public static var NIGHT:String = "n";
public static var THIS_MORNING:String = "This Morning";
public static var TODAY:String = "Today";
public static var TONIGHT:String = "Tonight";

private var date:Date;
private var dayOrNight:String;
private var timeOfDay:String;
```

4. Next, create the `updateTimeStr()` and `returnTwoChars()` methods. The `updateTimeStr()` method creates a string called `timeStr`, which simply contains the hours, minutes, date, month, and year. This is one of three variables that are subsequently added to an array; the other two are `timeOfDay` and `dayOrNight`. The array is set directly on the data object of the `TimeProxy` via `setData()`. This will allow data to be retrieved through the `getData()` call. The `returnTwoChars()` method is created to ensure that the time values returned from the `Date` object return two characters (see Listing 11-42).



Available for download on Wrox.com

LISTING 11-42: The updateTimeStr and returnTwoChars methods for TimeProxy

```
private function updateTimeStr():Void {
      var hrs:String = returnTwoChars(date.getHours());
      var mins:String = returnTwoChars(date.getMinutes());
      var day:String = returnTwoChars(date.getDate());
      var month:String = returnTwoChars((date.getMonth() + 1));
      var year:String = date.getFullYear().toString();
      var timeStr:String = hrs + ":" + mins + ", " + day + "/" + month + "/" + year;

      setData([timeOfDay, dayOrNight, timeStr]);
      sendNotification(ApplicationFacade.TIME_UPDATED, getData());
}

private function returnTwoChars(num:Number):String {
      var chars:String;
      if(num < 10){
            chars = "0" + num.toString();
      } else {
            chars = num.toString();
      }

      return chars;
}
```

www.it-ebooks.info

5. Create the `isDayOrNight()` method. Here, the `timeOfDay` variable is set, and this can be determined from the number of hours returned by calling the `getHours()` method on the `Date` instance called `date`. Within the method, an `if` statement should define morning as anytime between 4 A.M. and 12 P.M., and `timeOfDay` is set to `THIS_MORNING`. Daytime should be defined as anytime between 12 P.M. and 8 P.M., where `timeOfDay` is then set to `TODAY`. Finally, night is anytime between 8 P.M. and 4 A.M., and `timeOfDay` is then set to `TONIGHT`. The `updateTimeStr()` method is then called to set the `data` object for the proxy (see Listing 11-43).



LISTING 11-43: The isDayOrNight method for TimeProxy

Available for download on Wrox.com

```
private function isDayOrNight():Void {

        var hrs:Number = date.getHours();

        if (hrs < 20 && hrs > 12) {
              dayOrNight = DAY;
              timeOfDay = TODAY;
        } else if (hrs > 4 && hrs < 12) {
              dayOrNight = DAY;
              timeOfDay = THIS_MORNING;
        } else {
              dayOrNight = NIGHT;
              timeOfDay = TONIGHT;
        }

        updateTimeStr();
}
```

6. Lastly update the `onRegister()` method to instantiate the `date`, and then call the `isDayOrNight()` method (see Listing 11-44).



LISTING 11-44: The onRegister method for TimeProxy

Available for download on Wrox.com

```
public function onRegister():Void {
        date = new Date();
        isDayOrNight();
}
```

This completes the `TimeProxy`. Now take a look at the final proxy utilized in the Weather Client, the `FSCommandsProxy`.

FSCommandsProxy.as

When compared to earlier chapters, the `onRegister()` method for `FSCommandsProxy` has an additional `fscommand2()` call, called `DisableKeypadCompatibilityMode`.

www.it-ebooks.info

`DisableKeypadCompatibilityMode` is implemented here to remove the on-screen touch keypad, which is added to S60 5th edition touch devices, such as the Nokia N97 and Nokia 5800 XpressMusic. The `fscommand2()` call is also present; it means that the resolution you developed for is also reflected on the device (see Listing 11-45).



LISTING 11-45: Disabling the keypad compatibility mode in FSCommandsProxy

```
import org.puremvc.as2.interfaces.IProxy;
import org.puremvc.as2.patterns.proxy.Proxy;

class com.wrox.chapter11.model.FSCommandsProxy extends Proxy implements IProxy {

      public static var NAME:String = "FSCommandsProxy";

      private var fscommand2:Function;

      public function FSCommandsProxy(data:Object) {
            super(NAME, data);
      }

      public function onRegister():Void {
            fscommand2("DisableKeypadCompatibilityMode");
            fscommand2("FullScreen", true);
            fscommand2("SetQuality", "high");
      }
}
```

When you're ready to test the application on a device, before doing so try removing this line and you will notice the touch keypad appears, reducing the application's real estate. This is something that can't be tested in Device Central CS4.

Later you will also look at the mediators that register the notification names we have detailed up to now and see how they are handled. Having covered each of the proxy classes in detail, the next few sections focus on the commands used in the application.

Creating the Weather Client's Controller

In this section you'll go through the commands used for the controller in the Weather Client (see Figure 11-7).

www.it-ebooks.info

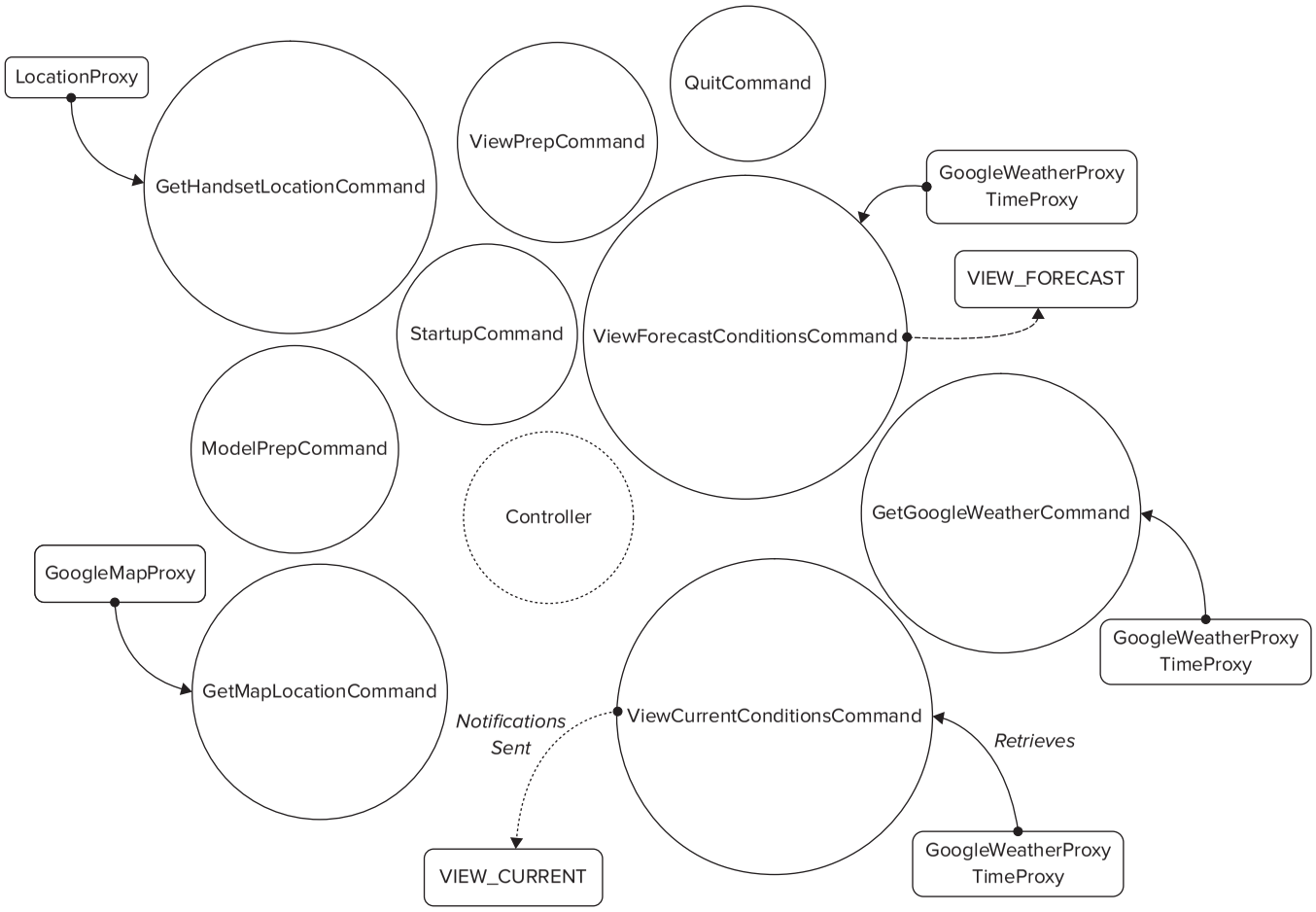


FIGURE 11-7

www.it-ebooks.info

There are five new commands discussed in this section; each simply overrides the `execute()` method to carry out a request for the application.

In this section you'll cover the commands responsible for:

- Preparing the application
- Getting the weather
- Retrieving the location
- Displaying the weather

The Project panel in Figure 11-8 shows each of the files under the controller directory.



FIGURE 11-8

Preparing the Application

Next you'll cover the preparation of the PureMVC application.

StartupCommand.as

The `StartupCommand` extends `MacroCommand` and registers two commands with the controller via the `initializeMacroCommand()` method: `ModelPrepCommand` and `ViewPrepCommand` (see Listing 11-46).

LISTING 11-46: The initializeMacroCommand method for StartupCommand

Available for download on Wrox.com

```
public function initializeMacroCommand():Void {
        addSubCommand(new ModelPrepCommand());
        addSubCommand(new ViewPrepCommand());
}
```

ModelPrepCommand.as

The `ModelPrepCommand` is responsible for registering the proxy classes for the Weather Client. The `execute()` method for `ModelPrepCommand` registers five proxies with the model (see Listing 11-47).

LISTING 11-47: The execute method for ModelPrepCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {
        facade.registerProxy(new FSCommandsProxy());
        facade.registerProxy(new GoogleMapProxy());
        facade.registerProxy(new GoogleWeatherProxy());
        facade.registerProxy(new TimeProxy());
        facade.registerProxy(new LocationProxy());
}
```

www.it-ebooks.info

ViewPrepCommand.as

The `ViewPrepCommand` registers six mediators with the view of the application (see Listing 11-48).



LISTING 11-48: The execute method for ViewPrepCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {
        var mc:MovieClip = MovieClip(note.getBody());
        facade.registerMediator(new OptionsMediator(mc));
        facade.registerMediator(new SoftKeyMediator(mc));
        facade.registerMediator(new SearchMediator(mc));
        facade.registerMediator(new CurrentConditionsMediator(mc));
        facade.registerMediator(new ForecastConditionsMediator(mc));
}
```

Getting the Weather

Retrieving weather data will simply require a call to the api, utilizing coordinates from the device.

GetGoogleWeatherCommand.as

The `GetGoogleWeatherCommand` is mapped to the `GET_GOOGLE_WEATHER` notification name, and when this command is executed, the `load()` method of `GoogleWeatherProxy` is called. This triggers the Google Weather API call, as discussed earlier. The retrieves the coordinates and address from the notification body. If you remember, these are sent from the `GoogleMapProxy` (see Listing 11-49).



LISTING 11-49: The execute method for GetGoogleWeatherCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {

        var coordinates:Object = note.getBody().Coordinates;
        var addressQuery:String = note.getBody().Address;

        var latitude:String = String(coordinates[0]).split(".").join("");
        var longitude:String = String(coordinates[1]).split(".").join("");

        var weatherQuery:String = ",,," + longitude + "," + latitude;
        var tProxy:TimeProxy = TimeProxy (facade.retrieveProxy(TimeProxy.NAME));
        var timeOfDayQuery:String = tProxy.getData()[0];
        var dayOrNightQuery:String = tProxy.getData()[1];
        var dateStrQuery:String = tProxy.getData()[2];

        var gwProxy:GoogleWeatherProxy =
GoogleWeatherProxy(facade.retrieveProxy(GoogleWeatherProxy.NAME));
        gwProxy.load(weatherQuery, addressQuery, timeOfDayQuery, dayOrNightQuery,
dateStrQuery);

}
```

While this command uses the longitude and latitude coordinates as the weather query, the command could be modified to simply use one of the search types covered early in the chapter.

Retrieving the Location

Next take a look at the commands responsible for retrieving a location.

www.it-ebooks.info

GetMapLocationCommand.as

The `GetMapLocationCommand` is mapped to the `GET_MAP_LOCATION` notification name. When the command is executed, the `getLocation()` method of the `GoogleMapProxy` is called, which triggers retrieval of the location specified by the user. The notification body, `note.getBody()`, is passed to the `locationQuery` variable, which is then supplied as the parameter for the `getLocation()` call on the proxy. You should also notice that the map proxy `GoogleMapProxy` is retrieved from the Facade and assigned to `mProxy` (see Listing 11-50).



LISTING 11-50: The execute method of GetMapLocationCommand

```
public function execute(note:INotification):Void {
      var locationQuery:String = String(note.getBody());
      var mProxy:GoogleMapProxy =
GoogleMapProxy(facade.retrieveProxy(GoogleMapProxy.NAME));
      mProxy.getLocation(locationQuery);
}
```

GetHandsetLocationCommand.as

The `GetHandsetLocationCommand` is mapped to the `GET_HANDSET_LOCATION` notification name and uses `initialize()` method of `LocationProxy` to call to initialize the Nokia S60 Location Service. The `LocationProxy` is retrieved from the Facade by using the `retrieveProxy()` method, then finally `initialize()` is called on the `lProxy` instance (see Listing 11-51).



LISTING 11-51: The execute method for GetHandsetLocationCommand

```
public function execute(note:INotification):Void {
      var lProxy:LocationProxy = LocationProxy(
facade.retrieveProxy(LocationProxy.NAME));
      lProxy.initialize();
}
```

Displaying the Weather

Next take a look at the commands for displaying weather information.

ViewCurrentConditionsCommand.as

The `ViewCurrentConditionsCommand`, mapped to the `VIEW_CURRENT_CONDITIONS` notification name is responsible for responding to the user's call in the application to display the current weather conditions. In the `execute()` method, the command retrieves `GoogleWeatherProxy`, which is referenced as a local variable called `gwProxy` (see Listing 11-53). The aim of the command is to get the latest "weather data" that was loaded into the application, and then send the `WeatherVO` onto the view mediator via the `VIEW_CURRENT` notification name. You will notice here that the proxy `TimeProxy` is used. Here, this proxy data array is passed to the `timeOfDay` object. The first value is `timeOfDay[0]`, which is the time of day, and the second value is `timeOfDay[2]`, which is the actual date. These values are set on the `WeatherVO` retrieved from `GoogleWeatherProxy`. The `timeOfDay[0]` value is set on the `currentConditions.time_of_day` property of the `WeatherVO`., while `timeOfDay[2]` is set on the `date` property of the `WeatherVO` (see Listing 11-52).

www.it-ebooks.info



LISTING 11-52: The execute method for ViewCurrentConditionsCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {

      var gwProxy:GoogleWeatherProxy = GoogleWeatherProxy(facade.retrieveProxy(Googl
eWeatherProxy.NAME));
      var timeOfDay:Object = TimeProxy(facade.retrieveProxy(TimeProxy.NAME)).
getData();

      var wVO:WeatherVO = WeatherVO(gwProxy.getData());
      wVO.currentConditions.time_of_day = timeOfDay[0];
      wVO.date = timeOfDay[2];

      sendNotification(ApplicationFacade.VIEW_CURRENT, wVO);
}
```

ViewForecastConditionsCommand.as

The `ViewForecastConditionsCommand` is mapped to the notification name `VIEW_FORECAST_CONDITIONS` and runs the following `execute()` method. The command retrieves the weather data from the proxy `GoogleWeatherProxy`. The command sends the notification name `VIEW_FORECAST`, passing the data set in the proxy in the notification body (see Listing 11-53).



LISTING 11-53: The execute method for ViewForecastCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {
      var gwProxy:GoogleWeatherProxy =
GoogleWeatherProxy(facade.retrieveProxy(GoogleWeatherProxy.NAME));
      var wVO:WeatherVO = WeatherVO(gwProxy.getData());
      sendNotification(ApplicationFacade.VIEW_FORECAST, wVO);
}
```

QuitCommand.as

Again, the `QuitCommand` is used so that the user can quit the application. This should be familiar to you (see Listing 11-54).



LISTING 11-54: The execute method for QuitCommand

Available for download on Wrox.com

```
public function execute(note:INotification):Void {
      fscommand("Quit");
}
```

Next, we'll explore the view elements.

Exploring the View

In this section you'll explore the view elements used for the Weather Client application (see Figure 11-9).

www.it-ebooks.info

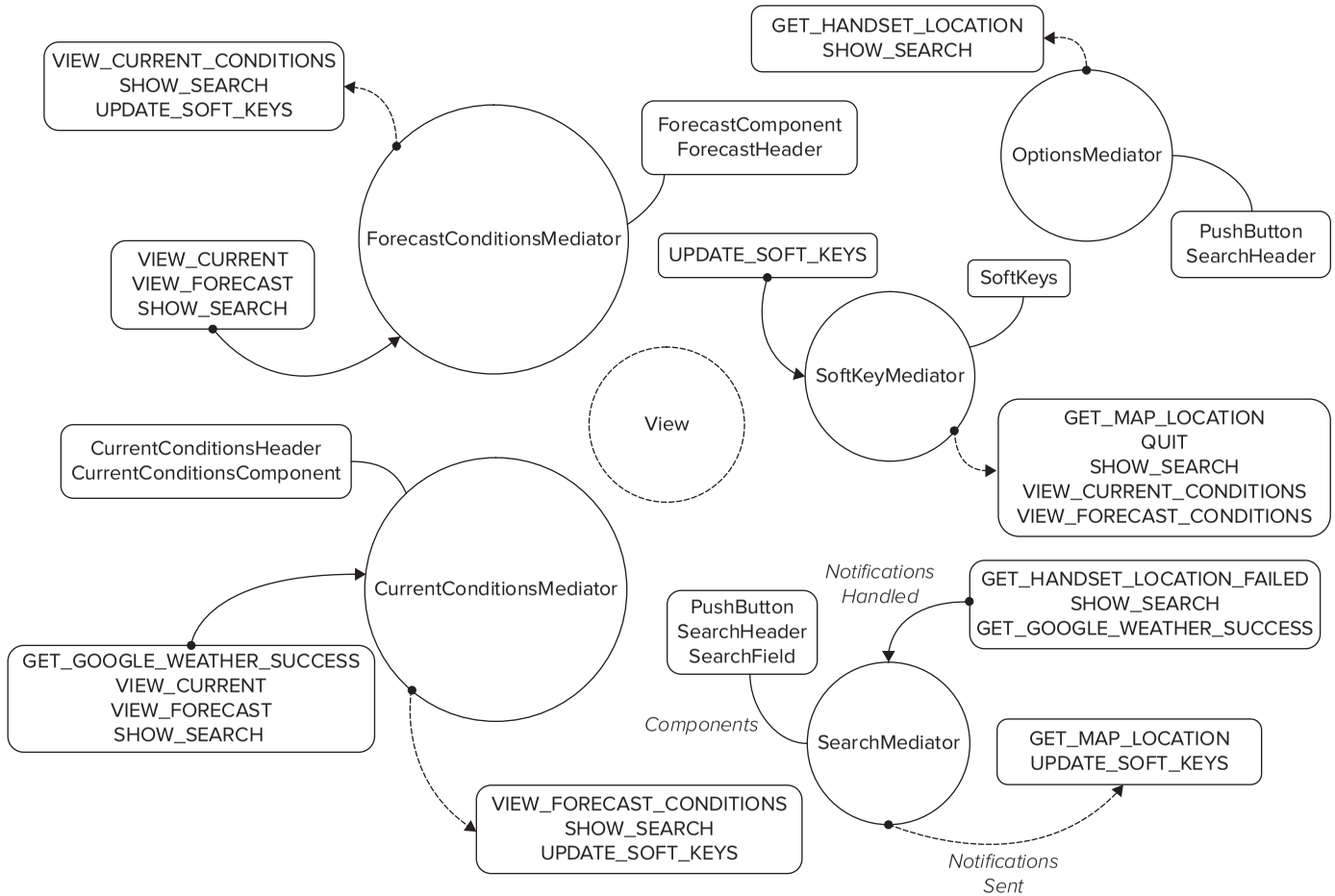


FIGURE 11-9

www.it-ebooks.info

In Figure 11-9 you can see the core mediators used in the Weather Client application.

The Project panel in Figure 11-10 shows each of the files under the view directory.

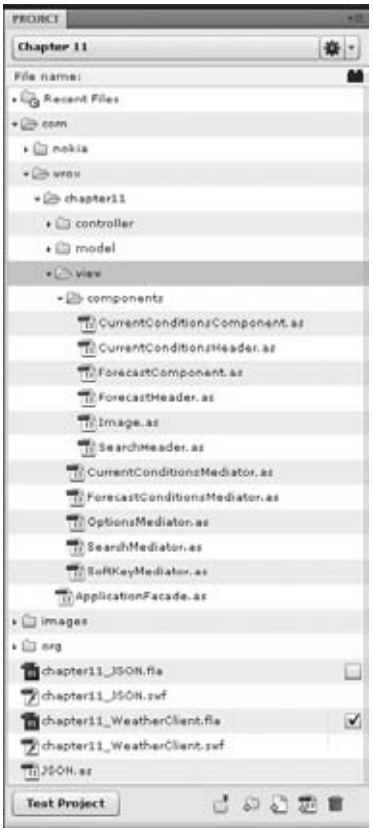


FIGURE 11-10

CurrentConditionsMediator.as

The method `listNotificationInterests()` has four notification names listed: `GET_GOOGLE_WEATHER_SUCCESS`, for when the weather results have been returned successfully; `SHOW_SEARCH`, for when the user decides to search for weather based on location; `VIEW_CURRENT`, for when the user selects the option to view the current weather data; and `VIEW_FORECAST`, for when the user selects to view the forecast weather data (see Listing 11-55).



LISTING 11-55: The listNotificationInterests method for CurrentConditionsMediator

Available for download on Wrox.com

```
public function listNotificationInterests():Array {
        return [ ApplicationFacade.GET_GOOGLE_WEATHER_SUCCESS,
                 ApplicationFacade.SHOW_SEARCH,
                 ApplicationFacade.VIEW_CURRENT,
                 ApplicationFacade.VIEW_FORECAST ];
    }
```

www.it-ebooks.info

In the following snippet for the method, you can see that when the `VIEW_CURRENT` and `GET_GOOGLE_WEATHER_SUCCESS` notifications are sent, the `showCurrentConditions()` method is called by the mediator. You should also note that, when the `VIEW_FORECAST` and `SHOW_SEARCH` notifications are sent, the mediator's response is to call the `removeCurrentConditions()` method (see Listing 11-56).



LISTING 11-56: The handleNotification method for CurrentConditionsMediator

```
public function handleNotification(note:INotification):Void {
      switch (note.getName()) {
            case ApplicationFacade.GET_GOOGLE_WEATHER_SUCCESS:
                  showCurrentConditions(note);
                  updateSoftKeys();
            break;
            case ApplicationFacade.SHOW_SEARCH:
                  removeCurrentConditions();
            break;
            case ApplicationFacade.VIEW_CURRENT:
                  showCurrentConditions(note);
                  updateSoftKeys();
                  break;
            case ApplicationFacade.VIEW_FORECAST:
                  removeCurrentConditions();
            break;
      }
}
```

The notification `note` is supplied as a parameter for the `showCurrentConditions()` method. It then uses `note.getBody()` to set the data provider of the header on the mediator `conditionsHeader` and updates the soft-keys component (see Listing 11-57).



LISTING 11-57: The showCurrentConditions method for CurrentConditionsMediator

```
private function showCurrentConditions(note:INotification):Void {

      var wVO:WeatherVO = WeatherVO(note.getBody());
      var cVO:ConditionVO = wVO.currentConditions;

      conditionsHeader = new CurrentConditionsHeader(_component);
      conditionsHeader.dataProvider = [wVO.date, wVO.address, cVO.icon_sml];

      currentConditions = new CurrentConditionsComponent(_component);
      currentConditions.dataProvider = [cVO.time_of_day, cVO.condition, cVO.temp_f,
cVO.icon_lrg];

}
```

The `updateSofyKeys()` method, which is called after the mediator shows the current conditions, basically updates the left soft key so that the "Search" is visible, and updates the right soft key so that the "Forecast" is also visible. The `SHOW_SEARCH` notification is sent from "Search", while `VIEW_FORECAST_CONDITIONS`, is sent from the "Forecast" (see Listing 11-58).

www.it-ebooks.info



LISTING 11-58: The updateSoftKeys method for CurrentConditionsMediator

Available for download on Wrox.com

```
private function updateSoftKeys():Void  {
      var msk:Object = {label:"", note:""};
      var lsk:Object = {label:" Search ", note:ApplicationFacade.SHOW_SEARCH};
      var rsk:Object = {label:" Forecast ",
note:ApplicationFacade.VIEW_FORECAST_CONDITIONS};
      sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
}
```

The final function in the `CurrentConditionsMediator` is the `removeCurrentConditions()` method, where each of the components is destroyed and effectively removed from view (see Listing 11-59).



LISTING 11-59: The removeCurrentConditions method for CurrentConditionsMediator

Available for download on Wrox.com

```
private function removeCurrentConditions():Void {
      conditionsHeader.destroy();
      currentConditions.destroy();
}
```

ForecastConditionsMediator.as

If you open the `ForecastConditionsMediator` class, you'll see that it uses two bespoked components, a forecast header component called `ForecastHeader`, which simply displays the location and date, and a forecast component called `ForecastComponent`, which is used to render weather data for the next four days (see Listing 11-60).



LISTING 11-60: The ForecastHeader and ForecastComponents in the ForecastConditionsMediator class

Available for download on Wrox.com

```
import com.wrox.chapter11.ApplicationFacade;
import com.wrox.chapter11.model.vo.WeatherVO;
import com.wrox.chapter11.model.vo.ConditionVO;
import com.wrox.chapter11.view.components.ForecastComponent;
import com.wrox.chapter11.view.components.ForecastHeader;
import mx.utils.Delegate;
import org.puremvc.as2.interfaces.IMediator;
import org.puremvc.as2.interfaces.INotification;
import org.puremvc.as2.patterns.mediator.Mediator;

class com.wrox.chapter11.view.ForecastConditionsMediator extends Mediator implements
IMediator {

      public static var NAME:String = "ForecastConditionsMediator";

      private var forecastHeader:ForecastHeader;
      private var forecast0:ForecastComponent;
      private var forecast1:ForecastComponent;
```

www.it-ebooks.info

```
        private var forecast2:ForecastComponent;
        private var forecast3:ForecastComponent;

        public function ForecastConditionsMediator(viewComponent:Object) {
              super(NAME, viewComponent);
        }

        public function getMediatorName():String {
              return ForecastConditionsMediator.NAME;
        }
```

The mediator lists three notification for various interests: `VIEW_CURRENT`, `VIEW_FORECAST`, and `SHOW_SEARCH`. These notifications are sent in response to user navigation (see Listing 11-61).



LISTING 11-61: The listNotificationInterests method for ForecastConditionsMediator

Available for download on Wrox.com

```
public function listNotificationInterests():Array {
      return [ ApplicationFacade.VIEW_CURRENT,
               ApplicationFacade.VIEW_FORECAST,
               ApplicationFacade.SHOW_SEARCH ];
}
```

Two methods are used for handling the notification interests in `handleNotificationInterests()`: `showForecast()` and `removeForecast()`. When the `VIEW_FORECAST` notification is sent, the mediator calls the `showForecast()` method to display the forecast for the location, whether specified by the user or obtained through the S60 Location Service API. When either of the notifications `VIEW_CURRENT` or `SHOW_SEARCH` is sent, the mediator calls the `removeForecast()` method, which removes each component from the view (see Listing 11-62).



LISTING 11-62: The handleNotification method for ForecastConditionsMediator

Available for download on Wrox.com

```
public function handleNotification(note:INotification):Void {
      switch (note.getName()){
            case ApplicationFacade.VIEW_CURRENT:
                  removeForecast();
            break;
            case ApplicationFacade.SHOW_SEARCH:
                  removeForecast();
            break;
            case ApplicationFacade.VIEW_FORECAST:
                  showForecast(note);
                  updateSoftKeys();
            break;
      }
}
```

The `showForecast()` method uses the notification body and sets the `WeatherVO` for the `ForecastHeader` component. Here, the forecast header component is created and the data is set on the instance `forecastHeader.dataProvider`, which is an array. This consists of the "Forecast"

www.it-ebooks.info

text, the address for the weather information, `wVO.address`, and finally the small image path value for the current weather condition, which is `wVO.currentConditions.icon_sml`. The data for the forecast conditions is also created in a `for` loop, which iterates through the weather conditions saved in the `wVO.forecastConditions` array. Four forecast components are also instantiated and added to the view. Each has its `_y` position and `dataProvider` set. The `dataProvider` is an array consisting of `ConditionVO` data properties, including: `cVO.day_of_week`, `cVO.condition`, `cVO.high`, `cVO.low`, and `cVO.icon_sml`. All of these properties were covered earlier (see Listing 11-63).



LISTING 11-63: The showForecast method for ForecastConditionsMediator

Available for download on Wrox.com

```
public function showForecast(note:INotification):Void {

      var cVO:ConditionVO;
      var wVO:WeatherVO = WeatherVO(note.getBody());
      var thisY:Number = 125;

      forecastHeader = new ForecastHeader(_component);
      forecastHeader.dataProvider = ["Forecast", wVO.address, wVO.currentConditions.
icon_sml];

      for (var i:Number = 0; i < 4; i++){

            var cVO:ConditionVO = ConditionVO(wVO.forecastConditions[i]);

            this["forecast" + i] = new ForecastComponent(_component, 40, thisY);
            this["forecast" + i]._y = thisY;
            this["forecast" + i].dataProvider = [cVO.day_of_week, cVO.condition,
cVO.high, cVO.low, cVO.icon_sml];

            thisY += 100;
      }
}
```

The `removeForecast()` method simply removes the forecast header and the four forecast components by calling their `destroy()` methods (see Listing 11-64).



LISTING 11-64: The removeForecast method for ForecastConditionsMediator

Available for download on Wrox.com

```
public function removeForecast():Void {
      forecastHeader.destroy();
      forecast0.destroy();
      forecast1.destroy();
      forecast2.destroy();
      forecast3.destroy();
}
```

After calling the `showForecast()` method, the `updateSoftKeys()` method is called. This allows the user to navigate back to the "Current" weather conditions view or navigate to the "Search" section on the application (see Listing 11-65).

www.it-ebooks.info



LISTING 11-65: The updateSoftKeys method for ForecastConditionsMediator

```
private function updateSoftKeys():Void {
      var msk:Object = {label:"", note:""};
      var lsk:Object = {label:" Search ", note:ApplicationFacade.SHOW_SEARCH};
      var rsk:Object = {label:" Current ",
note:ApplicationFacade.VIEW_CURRENT_CONDITIONS};
      sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
}
```

Finally, let's take a look at the search mediator used in the application.

SearchMediator.as

The search mediator has three private properties, including a submit button for the search, called `searchBtn`; a header for the view, called `searchHeader`; and a search field `searchField` for the user's input. These components are collectively placed in the mediator (see Listing 11-66).



LISTING 11-66: The components used in SearchMediator

```
private var searchButton:MovieClip;
private var searchHeader:SearchHeader;
private var searchField:SearchField;
```

The `SearchMediator` registers to listen to three notifications via the `listNotificationInterests` method: `GET_HANDSET_LOCATION_FAILURE`, for when the application fails to retrieve the GPS location; `SHOW_SEARCH`, for when the users want to use the search; and `GET_GOOGLE_WEATHER_SUCCESS`, for when the weather has been retrieved through the `GoogleWeatherProxy` (see Listing 11-67).



LISTING 11-67: The listNotificationInterests method for SearchMediator

```
public function listNotificationInterests():Array {
      return [ ApplicationFacade.GET_HANDSET_LOCATION_FAILURE,
               ApplicationFacade.SHOW_SEARCH,
               ApplicationFacade.GET_GOOGLE_WEATHER_SUCCESS ];
}
```

The `handleNotification()` method shows how the `SearchMediator` handles notifications (see Listing 11-68).



LISTING 11-68: The handleNotification method for SearchMediator

```
public function handleNotification(note:INotification):Void {
      switch (note.getName()) {
            case ApplicationFacade.GET_HANDSET_LOCATION_FAILURE:
                  showSearchPanel(note);
                  updateSoftKeys();
```

continues

www.it-ebooks.info

LISTING 11-68 *(continued)*

```
                break;
                case ApplicationFacade.SHOW_SEARCH:
                      showSearchPanel(note);
                      updateSoftKeys();
                break;
                case ApplicationFacade.GET_GOOGLE_WEATHER_SUCCESS:
                      searchText.removeTextField();
                      searchHeader.destroy();
                      removeMovieClip(searchButton);
                default:
                break;
          }
}
```

From the `handleNotifications()` method, you'll be able to see that when the `GET_HANDSET_LOCATION_FAILURE` and `SHOW_SEARCH` notification names are sent, the `showSearchPanel()` method is called. Following this, the familiar `updateSoftKeys()` method is called and updates the `SoftKeysMediator` to "Quit" the application when the left soft key is pressed (see Listing 11-69).



Available for download on Wrox.com

LISTING 11-69: The showSearchPanel and updateSoftKeys methods for SearchMediator

```
public function showSearchPanel(note:INotification):Void {

      headeader = new SearchHeader(_component);
      searchHeader.dataProvider = ["Search", "Use the textfield to enter a place.."];

      var format:TextFormat = new TextFormat();
      format.font = "Arial";
      format.size = 18;
      format.color = 0x000000;

      _component.createTextField("sText", _component.getNextHighestDepth(), 26, 100,
200, 30);

      searchTxt = new TextField();
      searchTxt = _component.sText;
      searchTxt.embedFonts = true;
      searchTxt.type = "input";
      searchTxt.background = true;
      searchTxt.borderColor = 0xff0000;
      searchTxt.multiline = false;
      searchTxt.setNewTextFormat(format);

      searchBtn = _component.attachMovie("PushButton", "searchBtn", _component.
getNextHighestDepth() );
      searchBtn._x = 20;
```

www.it-ebooks.info

```
        searchBtn._y = 140;
        searchBtn._labelText = "Search";
        searchBtn.onSelect = Delegate.create(this, onSubmit);

    }

    private function updateSoftKeys():Void  {
        var msk:Object = {label:"", note:""};
        var lsk:Object = {label:" Quit ", note:ApplicationFacade.QUIT};
        var rsk:Object = {label:"", note:""};
        sendNotification(ApplicationFacade.UPDATE_SOFT_KEYS, [lsk, msk, rsk]);
    }
```

When the `showSearchPanel()` method is called it ensures the view displays the search field, search button, and header in the application. When the search button `searchBtn` is selected the `onSubmit()` method is called, which ensures that there is text entered in the search field before sending the `GET_MAP_LOCATION` notification (see Listing 11-70).



LISTING 11-70: The onSubmit method for SearchMediator

```
public function onSubmit():Void {
        if (searchTxt.text != "") {
                sendNotification(ApplicationFacade.GET_MAP_LOCATION, searchTxt.text);
        }
}
```

OptionsMediator.as

The final mediator used is `OptionsMediator`, which is actually the first mediator to be rendered in the application. The mediator serves to give the end user two options, with the ultimate aim of retrieving the weather. Because the `OptionsMediator` is the first mediator to be used, its `onRegister()` method draws the two `PushButton` instances, called `getWeatherBtn` and `searchWeatherBtn` (see Listing 11-71).



LISTING 11-71: The onRegister method for SearchMediator

```
public function onRegister():Void {

        getWeatherBtn = _component.attachMovie("PushButton", "getWeatherBtn", _
component.getNextHighestDepth() );
        getWeatherBtn._x = 0 + Stage.width/2 – getWeatherBtn._width/2;
        getWeatherBtn._y = 140;
        getWeatherBtn._labelText = "My Weather";
        getWeatherBtn.onSelect = Delegate.create(this, onGetWeather);

        searchWeatherBtn = _component.attachMovie("PushButton", "searchWeatherBtn",
_component.getNextHighestDepth() );
```

continues

www.it-ebooks.info

LISTING 11-71 *(continued)*

```
        searchWeatherBtn._x = 0 + Stage.width/2 – getWeatherBtn._width/2;
        searchWeatherBtn._y = 200;
        searchWeatherBtn._labelText = "Search";
        searchWeatherBtn.onSelect = Delegate.create(this, onSearch);

    }
```

Notice that each button has its own `onSelect()` method defined. The `onGetWeather()` method sends the `GET_HANDSET_LOCATION` notification, while `onSearch()` method sends the `SHOW_SEACH` notification (see Listing 11-72).



LISTING 11-72: The onGetWeather and onSearch methods for SearchMediator

Available for download on Wrox.com

```
public function onGetWeather():Void {
        searchWeatherBtn.removeMovieClip();
        getWeatherBtn.removeMovieClip();
        sendNotification(ApplicationFacade.GET_HANDSET_LOCATION);
}

public function onSearch():Void {
        searchWeatherBtn.removeMovieClip();
        getWeatherBtn.removeMovieClip();
        sendNotification(ApplicationFacade.SHOW_SEARCH);
}
```

Viewing the Application in Device Central

You've now covered the core classes and components of the Weather Client application. When you run the example in Device Central, you will be able to use the search feature to retrieve the weather forecast based on your input into the search field.

Because of the screen size targeted, the application should be tested with the Nokia 5800 XpressMusic device profile.

As discussed in the Chapter 10, on the Nokia Service API, in order to fully test the actual extended device features, you'll need to transfer the application's `.swf` file and resource files to a compatible device. However, earlier, in the discussion on the `LocationProxy`, you were showed how to get around the absence of the Location Service in providing the coordinates for a remote location.

When testing on the device, you will be restricted by the cross-domain security of the Google Weather API. You'll may need to make modifications so that the calls for the Google Weather API are handled by a server-side proxy. One suggestion would be to create a SWX API Service, which could handle all the processing of the results from both the Google Map API and Google Weather.

www.it-ebooks.info

SUMMARY

Although we're at the end, there are still lots of features that could be added to the Weather Client application. For instance, the background clip could change to display the day, afternoon, or night view, depending on the time. The weather icons could be animated. Or a simple wait indicator could be added to the view, to give users an indication of the progress of each request. So don't stop here; have a go at adding some features of your own.

In this chapter, you examined the code behind the Flash Lite Weather Client and learned about using the Google Maps and Google Weather API to retrieve data from the Google Weather. You learned how to integrate and utilize the Nokia Location Service API and also how to parse JSON.

In the next chapter, you take a look at Flash mobile content through the eyes of Sony Ericsson, with Project Capuchin.

www.it-ebooks.info

12

Using Sony Ericsson's Project Capuchin Platform Services API

WHAT'S IN THIS CHAPTER?

- Obtaining data via the accelerometer
- Searching for devices with Bluetooth
- Accessing a calendar and phone book
- Retrieving a device's folder paths
- Formatting currency based on locale
- Utilizing API Test data in Device Central

In this chapter you will learn about how you can extend the features of your Flash Lite mobile applications, targeting Sony Ericsson devices using the Project Capuchin Service API (`http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp`).

You will be provided with snippets and code examples that you can use in the development of your applications, while exploring some of the APIs.

The Project Capuchin Platform Services API library, which is referenced in this chapter, can be found at the Sony Ericsson Developers website at `http://developer.sonyericsson.com`. There you'll also find demos of other mobile applications and tutorials and another cool community of developers.

www.it-ebooks.info

PROJECT CAPUCHIN PLATFORM SERVICES OVERVIEW

> *"Project Capuchin is a bridging technology between Java ME and Flash Lite making it possible to take advantage of Flash's strengths in fast UI deployment and well established designer tools, meanwhile Java's strengths are in services, security and a well developed distribution infrastructure. Project Capuchin makes it possible to create Java applications where some or all UI components are defined in Flash Lite and where all services can be accessed through Java."*
>
> Sony Ericsson Developers Website

Project Capuchin effectively provides another way for you to extend the Flash Lite player for Sony Ericsson supported devices. While some of the APIs have functionality that is similar to that of the APIs available for supported Nokia devices, there are fundamental differences in the way they are used and implemented, as you will learn in this chapter.

To use them requires running several extension `.mxp` files through the Adobe Extension Manager. Components and relevant API classes are then integrated into Flash CS4 and can be referenced in AS. The set of APIs is available from the service MXP section of the Sony Ericsson developer website.

Project Capuchin API Services

At the time of writing, there is a total of 11 core Project Capuchin Service APIs open for Flash Lite developers to integrate into their projects. Here's a list and brief description of the functionality you can obtain with each:

- **Accelerometer** — Detect and respond to device orientation
- **Bluetooth** — Send and receive data through device connections
- **Calendar** — Create and save calendar dates
- **Contacts** — Access and control of a device's address book
- **File** — Access and manage files on a device's file system
- **I18n** — Format data, based on locale
- **Location** — Access to the GPS functionality of a device
- **Messaging** — Send and receive SMS and MMS messages
- **Multimedia** — Play and record audio files
- **Persistency** — Save and retrieve application data
- **Radio** — Control the radio on a device

In this chapter you'll see an overview of each service and take an in-depth look at the API services for Accelerometer, Bluetooth, Calendar, Contacts, Internationalization, Messaging, Persistency, and Radio, allowing you to explore the excellent resource of Sony Ericsson.

www.it-ebooks.info

How to Use the API Methods

In order to use these services in your Flash Lite application for Sony Ericsson devices, you will need to install a few extensions into Flash CS4 using the Adobe Extension Manager:

1. Download the Core Project Capuchin MXP Service from the Sony Ericsson developer website at `http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp`.
2. From the same site, you should be able to download each of the previously mentioned MXP Service APIs. You will need to install each of these individually using the Adobe Extension Manager.



To use any of the Service APIs, you must also install the Core Project Capuchin MXP file.

Once they are installed, you should be able to see each of the service components in the Components panel (see Figure 12-1).

The next section of this chapter details each of the services and how you can use them.



FIGURE 12-1

USING ACCELEROMETER

This API allows you to retrieve the axes' values and the device orientation from the Accelerometer on the device, which you can use in a number of ways to enhance user interaction and user experience for your applications.

API Features

There are eight methods associated with the Accelerometer service in total, all of which you will need to use to implement the service correctly:

- `addDataReceivedListener()` — Used to register a listener for receiving updates from the accelerometer
- `addOrientationChangedListener()` — Used to register a listener to receive notifications on the device orientation changes
- `close()` — Used to remove all listeners and close the connection to the accelerometer
- `getOrientation()` — Used to get the current device orientation
- `getValues()` — Used to get the current values from all accelerometer axes
- `init()` — Used to establish the connection to the accelerometer

www.it-ebooks.info

- `removeDataReceivedListener()` — Used to remove the listener for receiving new axis values from the accelerometer
- `removeOrientationChangedListener()` — Used to remove the listener for receiving notifications for device orientation changes

To demonstrate the feature of the Accelerometer API, you'll find a specially created class, called `AccelerometerAPI`, accompanying the associated files for the chapter, and an `.fla` file called `chapter12_Accelerometer.fla`, which will compile an example to show the results of the API service calls to display the orientation and axis values in text fields. When you run the example, you'll be able to use the "up" and "down" keys to retrieve these values from the API.

Initializing the Accelerometer API

To use the service, you need to ensure that you have imported the `Accelerometer` class. You then need to establish a connection from the Flash Lite player to the Accelerometer through the static `init()` call, passing the application reference for the call, which is usually the stage, or a `MovieClip` reference. The second parameter for the `init()` method should be a callback function.

Listing 12-1 shows the early stages of the `AccelerometerAPI` class, which we'll complete over the next few sections, so don't try running this quite just yet. You'll see that there are several private variables. The variable `connected` Boolean is a reference indicating whether or not the API has been initialized, `orientationTxt` is a `TextField` component where each of the orientation will be displayed, `axisTxt` will display the axis text, and `mc` is simply a reference to the target container for the example.

The constructor for `AccelerometerAPI` class calls the `init()` method, assigning a method called `initHandler()` as the callback function. Here, both text fields are also added to the stage, and then finally a `Key` listener is initialized to handle input from the user, via the `onKeyDownHandler()` method. Both handlers will be covered shortly.



LISTING 12-1: Calling the init method in the Accelerometer API example

```
import com.sonyericsson.capuchin.accelerometer.Accelerometer;
import mx.utils.Delegate;

class com.wrox.chapter12.AccelerometerAPI {

      private var connected:Boolean;
      private var mc:MovieClip;
      private var orientationTxt:TextField;
      private var axisTxt:TextField;

      public function AccelerometerAPI(target:MovieClip) {

            mc = target;
            Accelerometer.init(mc, Delegate.create(this, initHandler));
```

www.it-ebooks.info

```
            var depth1:Number = mc.getNextHighestDepth();
            mc.createTextField("orientationTxt", depth1, 5, 0, 240, 20);

            orientationTxt = new TextField();
            orientationTxt = mc.orientationTxt;
            orientationTxt.textColor = 0x000000;
            orientationTxt.text = "Orientation";

            var depth2:Number = mc.getNextHighestDepth();
            mc.createTextField("axisTxt", depth2, 5, 20, 240, 20);

            axisTxt = new TextField();
            axisTxt = mc.axisTxt;
            axisTxt.textColor = 0x000000;
            axisTxt.text = "Axis Values";

            var keyListener:Object = new Object();
            keyListener.onKeyDown = Delegate.create(this, onKeyDownHandler);
            Key.addListener(keyListener);

      }

      public function initHandler():Void { }

      public function onKeyDownHandler():Void { }

}
```



You will notice that all of the service methods for Project Capuchin are static, hence you don't need to create an instance of each service object to use it, unlike with the S60 Service API.

Before attempting to use other methods of the Accelerometer, you should check the `status` of the service connection. This can be done in the `initHandler()` callback method for the service.

Listing 12-2 shows the `initHandler()` for the `AccelerometerAPI`. The `status` and `owner` values parameters returned by the handler are used throughout the API. The `status` Boolean will define the success of the call, while the `owner` object returns the scope of the method call. The `connected` variable is set to true when the `connection` is successful.



LISTING 12-2: The initHandler method for Accelerometer API example

```
public function initHandler(owner:Object, status:Boolean):Void {
      if (status) {
            connected = true;
      }
}
```

www.it-ebooks.info

The orientation is defined by a number of axis coordinates on the mobile device: the x axis, y axis, and z axis. These property values are returned in a number of methods for the Accelerometer. You can retrieve these values from the device automatically or through user interaction.

Retrieving Updates Automatically

The simplest way to retrieve data from the Accelerometer is by using the listener methods `addDataReceivedListener()` and `addOrientationChangedListener()`. These methods broadcast data from the device's Accelerometer when it is updated, meaning you can retrieve updates automatically. After initializing the service, you just need to define callback handlers for the listener methods, passing a reference to the handler in the API call (see Listing 12-3).



LISTING 12-3: Calling the addDataReceivedListener and addOrientationChangedListener in the Accelerometer API example

```
public function initHandler(owner:Object, status:Boolean):Void {
       if (status) {
         connected = true;
         Accelerometer.addDataReceivedListener(mc, Delegate.create(this,
onDataReceivedHandler));
         Accelerometer.addOrientationChangedListener(mc, Delegate.create(this,
onOrientationHandler));
       }
}

public function onDataReceivedHandler():Void { }

public function onOrientationHandler():Void { }
```

Listing 12-4 shows the `addOrientationReceivedListener()` callback handler, `onOrientationHandler()`. Here, the text for the `orientationTxt` text field is set from the `orientation` value, one of the parameters returned in the callback.



LISTING 12-4: The onOrientationHandler method for Accelerometer API example

```
public function onOrientationHandler(owner:Object, status:Boolean, orientation:
String):Void {
      orientationTxt.text = "orientation = " + orientation;
}
```

The `orientation` parameter returned has four possible values `portrait`, `portrait_up_side_down`, `landscape_left`, and `landscape_right`. When you test the example in Device Central, the default value returned for `orientation` is `test string`; more on this later. Listing 12-5 shows the callback handler for `addDataReceivedListener()`, named `onDataReceivedHandler()`, which sets the text for the `axisTxt` text field.

www.it-ebooks.info



LISTING 12-5: The onDataReceivedHandler method for Accelerometer API example

```
public function onDataReceivedHandler(owner:Object, status:Boolean, axisX:Number,
axisY:Number, axisZ:Number, time:Number) {
      axisTxt.text = "time = " + time + ", axisX = " + axisX + ", axisY = " + axisY
+ ", axisZ = " + axisZ;
}
```

The `onDataReceivedHandler()` method returns each axis's value from the `Accelerometer`, (`axisX`, `axisY`, and `axisZ`) and also the timestamp, `time`, which represents the time at which the data was retrieved, allowing you to create some interesting speed calculations for consecutive updates.

Using the Remove Listener Methods

Use the `removeDataReceivedListener()` and `removeOrientationChangedListener()` methods when you no longer need to use the listener methods. As with the other methods for the API, you have to define a callback handler for each method. You can also use `close()` to remove all listeners and disconnect the Accelerometer.

Retrieving Device Orientation and Axis Values on Demand

The `getOrientation()` method specifically retrieves the current orientation of the device, while `getValues()` retrieves the current axis values for the device. You use either method when you want to retrieve the values through interaction with specific actions in your application such as a key press.

In the constructor of the `AccelerometerAPI` class, the `Key` class, Listing 12-6 shows how the callback methods for `getOrientation()` and `getValues()` are handled.



LISTING 12-6: The onKeyDownHandler, getOrientationHandler, and getValuesHandler methods for Accelerometer API example

```
public function onKeyDownHandler():Void {
      switch (Key.getCode()){
            case Key.UP:
                  Accelerometer.getOrientation(mc, Delegate.create(this,
getOrientationHandler));
            break;
            case Key.DOWN:
                  Accelerometer.getValues(mc, Delegate.create(this,
getValuesHandler));
            break;
      }
}

public function getOrientationHandler(owner:Object, status:Boolean, orientation:
String):Void {
      if (status) {
            orientationTxt.text = "orientation = " + orientation;
      }
}
```

continues

www.it-ebooks.info

LISTING 12-6 *(continued)*

```
public function getValuesHandler(owner:Object, status:Boolean, axisX:Number, axisY:
Number, axisZ:Number):Void {
      if (status) {
            axisTxt.text = "axisX = " + axisX + ", axisY = " + axisY + ", axisZ =
" + axisZ;
      }
}
```



Be aware that attempting to call either the `getOrientation()` *or* `getValues()` *method while the* `Accelerometer` *has a registered listener will result in an error being returned from these calls.*

Testing the API in Device Central

You have now covered each of the methods defined in the `AccelerometerAPI` class. If you open the `chapter12_Accelerometer.fla` file, on the first frame of the timeline, you'll see the following code snippet, which initializes the service:




```
import com.wrox.chapter12.AccelerometerAPI;

var accelerometer:AccelerometerAPI = new AccelerometerAPI(this);
```

code snippet chapter12_Accelerometer.fla

When you run the example in Device Central, you will see two text field labels set at the top left of the screen "Orientation" and "Axis Values." If you press "up," the orientation label changes to "orientation = test string," and if you press "down," the axis label changes to "axisX = 100, axisY = 100, axisZ = 100." These values are taken from a local data file called `Accelerometer.xml`. We'll take a look at this next.

Using the Accelerometer Test Data

If you remember, `getOrientationHandler()` and `getValuesHandler()` are both callback handlers that will have parameter values set from the device's Accelerometer when an application using the API runs on the device. Testing the methods of the API in Device Central can be accomplished via test data that is generated from a command in Flash Professional. In the Flash IDE, select Commands ➪ Prepare Capuchin test environment. For the Accelerometer API, the `Accelerometer.xml` file is created via this command. With the other APIs, you'll notice that test data is generated with the API name for the file name. When the application runs in Device Central, the `loadVariables()` method is called to load in the data from the `.xml` file.

www.it-ebooks.info

The following snippet shows a portion of the `Accelerometer.xml`, detailing some of the nodes and attributes:




```
<methods>
      <method method_name="getOrientation">
            <method_handler>
                  <handler_parameters>
                        <handler_parameter
                              handler_parameter_name="status"
                                handler_parameter_type="Boolean"
                                  is_array="false">
                              <items>
                                    <item>
                                          <value refers_to="status">
                                                <![CDATA[true]]>
                                          </value>
                                    </item>
                              </items>
                        </handler_parameter>
                        <handler_parameter
                              handler_parameter_name="orientation"
                                handler_parameter_type="String"
                                 is_array="false">
                              <items>
                                    <item>
                                          <value refers_to="orientation">
                                                <![CDATA[test string]]>
                                          </value>
                                    </item>
                              </items>
                        </handler_parameter>
                  </handler_parameters>
            </method_handler>
      </method>
</methods>
```

code snippet Accelerometer.xml

You should now be starting to picture how testing the API in Device Central works. First, when the test environment is created, the test data is generated. The nodes to look out for in the test data are the `handler_parameter` nodes. The snippet highlights the `getOrientation` method and defines the data for when the `getOrientation()` method is called. In the `handler_parameter` nodes, you can edit the values for the "status" and "orientation" parameters. The "status" parameter value is set to true, while the "orientation" parameter value is set to "test string," as mentioned earlier.

The point here is to gain an understanding of how you can use the test data when testing. This is common across all the API services, which generate their own set on APIs. To test you can edit the `.xml` file generated by the test environment and then restart or republish that `.fla` file to read the new data.

www.it-ebooks.info

USING BLUETOOTH

The Project Capuchin Bluetooth Service API provides you with several services using a device's Bluetooth capability, including creating a Bluetooth server to facilitate multiple device communication and sending files to other devices.

API Features

The Bluetooth (`http://www.bluetooth.com/`) wireless technology enables the exchange of data between electronic devices over short distances. You can use Bluetooth connections to send text, media, and data files such as calendar entries or business cards within a range of 33 feet (10 meters) and the connection is not just limited to mobile devices; you can interact with computers or potentially any device with that has Bluetooth capability.

There are numerous methods in Bluetooth API:

- `createServer()` — Used to create and initialize a named Bluetooth server
- `connectToServer()` — Used to connect to a specified Bluetooth server in the specified remote device
- `disconnectFromServer()` — Used to close a connection with the server
- `getNumberOfConnectedClients()` — Used to return a number of clients that are currently connected to the server exposed by local device
- `sendMessage()` — Used to send a message specified in message parameter
- `disconnectAllClients()` — Used to close connections with all clients connected to the Bluetooth server
- `getDeviceName()` — Used to return the name of the device
- `getBluetoothAddress()` — Used to get the Bluetooth address of the device
- `searchDevices()` — Used to search for remote devices
- `searchServer()` — Used to check if specified remote device exposes server of given name
- `isBluetoothOn()` — Used to check if a Bluetooth service is turned on and if device is visible
- `cancelDevicesSearch()` — Used to stop the process of searching remote devices in range
- `sendFile()` — Used to send a file specified by a URI parameter to defined remote device
- `addMessageReceivedListener()` — Used to register a listener for messages coming in from a remote device
- `removeMessageReceivedListener()` — Used to remove a listener for messages coming in from a remote device
- `addClientDisconnectedListener()` — Used to register a listener for client disconnection
- `removeClientDisconnectedListener()` — Used to remove a listener for client disconnection

www.it-ebooks.info

- `addServerDisconnectedListener()` — Used to register a listener for server disconnection
- `removeServerDisconnectedListener()` — Used to remove a listener for server disconnection
- `addClientConnectedToServerListener()` — Used to register a listener for new client connections
- `removeClientConnectedToServerListener()` — Used to remove a listener for new client connections

In this section you'll cover `getBluetoothAddress()`, `getDeviceName()`, `isBluetoothOn()`, and `searchDevices()`. You'll find sample code for these methods in the completed `BluetoothAPI` class, which we'll go through next. To use the Bluetooth API, you need ensure that you've imported the `Bluetooth` class and `BluetoothDevice` class (see Listing 12-7).



LISTING 12-7 Creating the Bluetooth API class example

```
import com.sonyericsson.capuchin.bluetooth.Bluetooth;
import com.sonyericsson.capuchin.bluetooth.datatypes.BluetoothDevice;
import mx.utils.Delegate;

class com.wrox.chapter12.BluetoothAPI {

      public function BluetoothAPI(){

      }

}
```

The `Bluetooth` class provides all the methods that are used for the API and `BlueToothDevice` is a data type representing a remote Bluetooth device. Each `BluetoothDevice` object has two string properties, one to represent the device name, called `deviceName`, and the other to represent the address, called `deviceAddress`.

Both properties can be accessed and set via their respective getter and setter methods on a `BluetoothDevice` instance. The `getDeviceName()` method retrieves the `deviceName`, whereas the `setDeviceName()` method allows you to modify `deviceName`. Likewise `getDeviceAddress()` retrieves `deviceAddress`, and `setDeviceAddress()` allows you to modify `deviceAddress`.

You should be aware that the majority of the methods for the API require the `BluetoothDevice` object.

Searching for Other Devices

The process for sending a message involves establishing a connection between two devices. In order to make a connection, the device sending the message needs to know the `deviceAddress` of the device receiving the message.

www.it-ebooks.info

Be aware that Bluetooth connectivity settings on a device can be managed by the end user. This is usually limited to being able to change the following:

- Connectivity — The end user can choose to turn Bluetooth on or off completely.
- Visibility — The end user can choose not to allow other devices to search for and find the device, usually by setting this option to "hide." The opposite is having the visibility set to "shown to all," which will allow the device to be found using a Bluetooth search. Some devices allow the end user to define a period of time during which the device is visible, setting it to be hidden after a timeout.
- Device Name — On the majority of mobile devices with Bluetooth support, the end user can define a name for the device so that it can be referenced via Bluetooth when the mobile device's Bluetooth visibility and connectivity settings are on.

Searching for another Bluetooth device when the target device visibility is "hidden" or the connectivity is "off" will mean you will not be able to find that device. The first step is, therefore, ensuring that these devices can actually "see" and connect to each other. The device sending the message needs to be on and using the `isBluetoothOn()` method to detect the Bluetooth connectivity status. In the `BluetoothAPI` example class, this is done in the constructor (see Listing 12-8).

In Listing 12-8 you'll see that there are a number of private variables declared and a few methods that are used in the example. The variables `nameList` and `addressList` are arrays to hold a reference to device names and device addresses, respectively. The `waitIndicator` and `softKeys` are MovieClip references to display their corresponding components, while the `deviceList` is a MovieClip to hold a dual-row list component, which displays device names and addresses. `btStatusTxt` and `sendStatusTxt` are text fields used to display the status of the Bluetooth *on* and file-sending methods. These will be updated throughout the example. The `mc` variable is a reference to the application container. The `draw()` method, called in the constructor, adds two of the components to the application when it is run. The `deviceList` has its `onSelect()` event handler delegated to a `sendFile()` method, while the `softKeys` component has its `onSoftKeyDown()` event delegated to the `search()` method. Both of these will be covered later.



Available for download on Wrox.com

LISTING 12-8: Calling isBluetoothOn method in the for BlueTooth API example

```
import com.sonyericsson.capuchin.bluetooth.Bluetooth;
import com.sonyericsson.capuchin.bluetooth.datatypes.BluetoothDevice;
import mx.utils.Delegate;

class com.wrox.chapter12.BluetoothAPI {

      private var nameList:Array;
      private var addressList:Array;
      private var waitIndicator:MovieClip;
      private var deviceList:MovieClip;
      private var softKeys:MovieClip;
      private var btStatusTxt:TextField;
```

www.it-ebooks.info

```
        private var sendStatusTxt:TextField;
        private var mc:MovieClip;

        public function BluetoothAPI(stageReference:MovieClip){
              mc = stageReference;
              Bluetooth.isBluetoothOn(mc, Delegate.create(this, isBluetoothOnHandler));
              draw();
        }

        private function isBluetoothOnHandler():Void { }

        private function draw():Void {
              deviceList = mc.attachMovie("ListTwoRowWithIcon", "deviceList", 0);
              deviceList._y = 40;

              softKeys = mc.attachMovie("SoftKeys", "softKeys", 1);
              softKeys._isLSKEnabled = false;
              softKeys._LSK = "";
              softKeys._isRSKEnabled = false;
              softKeys._RSK = "";
              softKeys._MSK = "SEARCH";
              softKeys.onSoftKeyDown = Delegate.create(this, search);

              mc.createTextField("btStatusTxt", 2, 5, 0, 240, 20);
              btStatusTxt = new TextField();
              btStatusTxt = mc.btStatusTxt;
              btStatusTxt.textColor = 0x000000;
              btStatusTxt.text = "Is on?";

              mc.createTextField("sendStatusTxt", 3, 5, 20, 240, 20);
              sendStatusTxt = new TextField();
              sendStatusTxt = mc. sendStatusTxt;
              sendStatusTxt.textColor = 0x000000;
              sendStatusTxt.text = "File sent?";

        }

        private function sendFile():Void { }

        private function search():Void { }

  }
```

The `isBluetoothOnHandler()` method returns three parameter values: the `owner` and `status`, as covered earlier, and `result`. The `result` of the method is a Boolean. If the `status` and `result` are both true, then the device name and device address can be retrieved on the device that is calling the method. For this, the static `getDeviceName()` and `getBluetoothAddress()` methods of the Bluetooth Service API are called (see Listing 12-9).

www.it-ebooks.info



LISTING 12-9 The isBluetoothOnHandler, method for BlueTooth API class example

```
private function isBluetoothOnHandler(owner:Object, status:Boolean, result:
Boolean):Void {
      if (status && result) {
            Bluetooth.getDeviceName(owner, Delegate.create(this, onGetDeviceName));
            Bluetooth.getBluetoothAddress(owner, Delegate.create(this,
onGetBluetoothAddress));
            btStatusTxt.text = "Bt is on";
      } else {
            btStatusTxt.text = "Bt is off";
      }
}

private function onGetDeviceName(owner:Object, deviceName:String):Void {
      btStatusTxt.text += ", name is " + deviceName;
}

private function onGetBluetoothAddress(owner:Object, deviceAddress:String):Void {
      btStatusTxt.text += ", address is " + deviceAddress;
}
```

If the device has Bluetooth activated, it can search for other Bluetooth-enabled devices to send a file to. The `search()` method, assigned to the `softKeys` component, invokes the static `searchDevices()` method of the Bluetooth Service API. This is assigned the callback method `onSearchDevices()`, which we'll cover shortly. The `waitIndicator` is also displayed while the search is taking place via the `showIndicator()` method (see Listing 12-10).



LISTING 12-10: The search, showIndicator and hideIndicator methods for the BlueTooth API example class

```
private function search():Void{
      showIndicator();
      Bluetooth.searchDevices(mc, Delegate.create(this, onSearchDevices));
}

private function onSearchDevices():Void {}

private function showIndicator():Void {
      waitIndicator = mc.attachMovie("WaitIndicator", "waitIndicator", 4);
      waitIndicator._shape = "circle";
      waitIndicator._x = 100;
      waitIndicator._y = 200;
}

private function hideIndicator():Void {
      waitIndicator.removeMovieClip();
}
```

The `onSearchDevices()` method has three parameter values set: the `owner` and `status`, and the `remoteDevices` parameter, which contains an array of remote devices, each with a `BluetoothDevice`

www.it-ebooks.info

data type. So once retrieved by iterating through the remote devices, the `deviceName` and `deviceAddress` properties can be saved to the `nameList` and `addressList` arrays. These are then added to the dual row list component `deviceList` using the `addItems()` method (see Listing 12-11).



LISTING 12-11: The onSearchDevices method for the BlueTooth API

```
public function onSearchDevices(owner:Object, status:Boolean, remoteDevices:Array):
Void {
      if(status) {
            nameList = [];
            addressList = [];

            for (var i:Number = 0; i< remoteDevices.length; i++) {
                  nameList.push(remoteDevices[i].deviceName);
                  addressList.push(remoteDevices[i].deviceAddress);
            }

            hideIndicator();

            deviceList.addItems(nameList, addressList);

            softKeys._MSK = "SEND FILE";
            softKeys.onSoftKeyDown = Delegate.create(this, sendFile);
      }
}
```

In Listing 12-12 you'll also notice that the `onSoftKeyDown()` method is redefined for the `softKeys` component, so that when the key is pressed, the `sendFile()` method is called. In the `sendFile()` method, the `deviceAddress` for sending the file to is specified.



LISTING 12-12: The sendFile method and onSendFile callback method for BluetoothAPI

```
private function sendFile():Void {
      var device:BluetoothDevice = addressList[deviceList.getFocusIndex()];
      var file:String = "file:////c:/picture/me.jpg";
      showIndicator();
      sendStatusTxt.text = "Sending = " + file + " to " + status.toString();
      Bluetooth.sendFile(this, Delegate.create(this, onSendFile), device, file);
}

private function onSendFile(owner:Object, status:Boolean):Void {
      hideIndicator()
      sendStatusTxt.text = "File was sent ok = " + status.toString();
}
```

The code to run the example is shown in the following snippet, which must be place on the first frame of the timeline:




```
import com.wrox.chapter12.BluetoothAPI;

var bluetooth:BluetoothAPI = new BluetoothAPI(this);
```

code snippet chapter12_Bluetooth.fla

www.it-ebooks.info

Using the Bluetooth Test Data

Again, when you run the test environment, the test data is generated to the `Bluetooth.xml` file, found in the same directory as the `.swf` file. The following snippet shows a portion of the `Bluetooth.xml`, detailing some of the nodes and attributes.

```
<items>
      <item>
            <value><![CDATA[SonyEricssonG750]]></value>
            <value><![CDATA[0000000DECAF]]></value>
      </item>
      <item>
            <value><![CDATA[SonyEricssonC905]]></value>
            <value><![CDATA[0000001DECAF]]></value>
      </item>
      <item>
            <value><![CDATA[SonyEricssonK770i]]></value>
            <value><![CDATA[0000002DECAF]]></value>
      </item>
      <item>
            <value><![CDATA[SonyEricssonK800i]]></value>
            <value><![CDATA[0000003DECAF]]></value>
      </item>
      <item>
            <value><![CDATA[SonyEricssonK750i]]></value>
            <value><![CDATA[0000004DECAF]]></value>
      </item>
</items>
```

USING CALENDAR

The Calendar Service API facilitates read and write capabilities for events in a device's calendar, from within the Flash Lite Player.

API Features

The service has six main methods:

- `addEvent()` — Create a new event
- `updateEvent()` — Update an existing event
- `removeEvent()` — Remove an existing event
- `removeAllEvents()` — Remove all events within a given date range
- `getView()` — Retrieve a list of events from date ranges
- `getEvents()` — Retrieve a list of events

For this API we simply demonstrate the usage of the `addEvent()` method, a useful feature that allows data within an application to be stored to an event in the default Calendar application on a device. With this feature you can extend applications, such as the TV Listings application covered earlier in the book, where the actual show time and date can be stored on the device as a reminder.

www.it-ebooks.info

The Calendar API has two main data types `CalendarEvent` class and `DateTime`. To use the Calendar Service, you need to add the core `Calendar` component to the `.fla` file and import the class along with `DateTime` and `CalendarEvent` classes.

The `chapter12_Calendar.fla` file contains the code that runs the example in the next section on the first frame of the timeline.

Creating a Calendar Event

Listing 12-13 shows the early stages of a sample class called `CalendarAPI`. The example shows how to create a calendar event on Valentine's Day, February 14, 2010, for a dinner reservation between 19:45 and 22:30. Here, there are several attributes for the event that have been defined: `summary`, `note`, `location`, `category`, and `privacyClass`. All are properties of the `CalendarEvent` class, and they are specified using each one's respective setter method.



LISTING 12-13: Creating a new calendar event in the Calendar API example

```
import com.sonyericsson.capuchin.calendar.Calendar;
import com.sonyericsson.capuchin.calendar.datatypes.CalendarEvent;
import mx.utils.Delegate;

class com.wrox.chapter12.CalendarAPI {

      private var statusTxt:TextField;
      private var valentinesDay:CalendarEvent;
      private var mc:MovieClip;

      public function CalendarAPI(target:MovieClip){

            mc = target;

            mc.createTextField("statusTxt", 1, 25, 25, Stage.width, Stage.height);
            statusTxt = new TextField();
            statusTxt = mc.statusTxt;
            statusTxt.text = "Output: " + newline;

            valentinesDay = new CalendarEvent();
            valentinesDay.setSummary("valentinesDay");
            valentinesDay.setNote("Dinner with Joanna at La Barbe");
            valentinesDay.setLocation("Reigate");
            valentinesDay.setCategory("Special Events");
            valentinesDay.setPrivacyClass("PRIVATE");
      }
}
```

To set the date for a calendar event, there are setter methods on the `CalendarEvent` instance for the start and end date properties of the `year`, `month`, `day`, `hour`, `minute`, and `second`. In Listing 12-14 you'll see that both the start date and end date for the event is defined, and the `setAlarm()` method is used to ensure that the user is reminded of the event 300 seconds (5 minutes) before it starts.

www.it-ebooks.info



LISTING 12-14: Specifying the start date and end date, and setting the alarm, for a calendar event

```
public function CalendarAPI(target:MovieClip){

      mc = target;

      mc.createTextField("statusTxt", 1, 25, 25, Stage.width, Stage.height);
      statusTxt = new TextField();
      statusTxt = mc.statusTxt;
      statusTxt.text = "Output: " + newline;

      valentinesDay = new CalendarEvent();
      valentinesDay.setSummary("valentinesDay");
      valentinesDay.setNote("Dinner with Joanna at La Barbe");
      valentinesDay.setLocation("Reigate");
      valentinesDay.setCategory("Special Events");
      valentinesDay.setPrivacyClass("PRIVATE");

      valentinesDay.setStartYear(2010);
      valentinesDay.setStartMonth(1);
      valentinesDay.setStartDay(14);
      valentinesDay.setStartHour(19);
      valentinesDay.setStartMinute(45);
      valentinesDay.setStartSecond(0);
      valentinesDay.setEndYear(2010);
      valentinesDay.setEndMonth(1);
      valentinesDay.setEndDay(14);
      valentinesDay.setEndHour(22);
      valentinesDay.setEndMinute(29);
      valentinesDay.setEndSecond(59);

      valentinesDay.setAlarm(300);

}
```

Having defined the core details for the event, it can now be added to the calendar via the `addEvent()` method. In Listing 12-15, you'll see that the `valentinesDay` object is supplied as a parameter to the `addEvent()` method, along with a reference to the `onAddEvent()` callback method. The callback method for `addEvent()` returns the `id` of the event added to the calendar, in addition to the `status` and `owner` values.



LISTING 12-15: Adding the event to the calendar

```
public function CalendarAPI(target:MovieClip){

      mc = target;

      mc.createTextField("statusTxt", 1, 25, 25, Stage.width, Stage.height);
      statusTxt = new TextField();
```

www.it-ebooks.info

```
        statusTxt = mc.statusTxt;
        statusTxt.text = "Text: " + newline;

        valentinesDay = new CalendarEvent();
        valentinesDay.setSummary("valentinesDay");
        valentinesDay.setNote("Dinner with Joanna at La Barbe");
        valentinesDay.setLocation("Reigate");
        valentinesDay.setCategory("Special Events");
        valentinesDay.setPrivacyClass("PRIVATE");

        valentinesDay.setStartYear(2010);
        valentinesDay.setStartMonth(1);
        valentinesDay.setStartDay(14);
        valentinesDay.setStartHour(19);
        valentinesDay.setStartMinute(45);
        valentinesDay.setStartSecond(0);
        valentinesDay.setEndYear(2010);
        valentinesDay.setEndMonth(1);
        valentinesDay.setEndDay(14);
        valentinesDay.setEndHour(22);
        valentinesDay.setEndMinute(29);
        valentinesDay.setEndSecond(59);

        Calendar.addEvent(this, Delegate.create(this, onAddEvent), valentinesDay);
    }

    private function onAddEvent(owner:Object, status:Boolean, id:String):Void {
        if(status == true){
            statusTxt.text += id;
        }
    }
```

The completed class will add a single calendar event to the Calendar application on a device. The following code snippet shows how to run the completed `CalendarAPI` example from the timeline.




```
import com.wrox.chapter12.CalendarAPI;

var calendar:CalendarAPI = new CalendarAPI(this);
```

code snippet chapter12_Calendar.fla

When you create a test environment and run the example in Device Central, you'll see the test data presented. In the next section you'll take a look at using the Contacts Service API.

USING CONTACTS

The Contacts API provides you with a way to manage contacts information from within your Flash Lite applications. You can use the API to create new or update existing contacts.

www.it-ebooks.info

API Features

The following list details the static methods that you can use in Contacts API version 1.0:

- `Contacts.addContact()` — Used to add a new entry to the device's phone book
- `Contacts.updateContact()` — Used to update an existing contact entry
- `Contacts.removeContact()` — Used to remove an existing entry from the contacts database
- `Contacts.getEntries()` — Used to retrieve one or more calendar entries
- `Contacts.removeAllContacts()` — Used to remove all the calendar entries specified

In this section you'll take an in-depth look at `addContact()` and `getEntries()`.

To use the Contacts API, you need to ensure that the `Contact` service component is on the stage of the `.fla` file. The `chapter12_Contacts.fla` file contains the code that runs the example in this section on the first frame of the timeline, which is referenced at the end of this section.

In addition to the `Contact` class, the API also has two associated data types: `ContactFilter` and `ContactEntry`. We'll take a look at using `ContactEntry` first.

Adding Contacts to a Phone Book

In Listing 12-16 you can see the start of a class, called `ContactsAPI`, being created. In the constructor you'll see that there is one text field created, called `statusTxt`, and four contacts defined. Here, the `setFirstName()` method on each `ContactEntry` instance is called to set the first name property of the contact. The `ContactEntry` can hold many details, including properties for home and work addresses, such as `homeNumber` and `workCity`, and personal details, such as `birthdayDay` and `pictureURL`. For this example just the `firstName` property is specified.



LISTING 12-16: Creating the Contacts API

```
import com.sonyericsson.capuchin.contacts.datatypes.ContactEntry;

class com.wrox.chapter12.ContactsAPI {

      private var statusTxt:TextField;
      private var mc:MovieClip;

      public function ContactsAPI(target:MovieClip){

            mc = target;

            mc.createTextField("statusTxt", 1, 25, 25, Stage.width, Stage.height);
            statusTxt = new TextField();
            statusTxt = mc.statusTxt;
            statusTxt.text = " Text: " + newline;

            var contact1:ContactEntry = new ContactEntry();
            contact1.setFirstName("Joseph");
```

www.it-ebooks.info

```
            var contact2:ContactEntry = new ContactEntry();
            contact2.setFirstName("Claire");

            var contact3:ContactEntry = new ContactEntry();
            contact3.setFirstName("Andrew");

            var contact4:ContactEntry = new ContactEntry();
            contact4.setFirstName("Wan Ping");

      }
}
```

To add each contact, the `addContact()` method needs to be called. The callback method `onAddContact()` is assigned to determine whether or not the call was successful and to retrieve the `id` parameter, which represents a unique ID for the contact. On successful addition, the `statusTxt` text field is updated with the `id` (see Listing 12-17).



LISTING 12-17: Adding a contact via the addContact method

```
import com.sonyericsson.capuchin.contacts.Contacts;
import com.sonyericsson.capuchin.contacts.datatypes.ContactEntry;
import mx.utils.Delegate;

class com.wrox.chapter12.ContactsAPI {

      private var statusTxt:TextField;
      private var mc:MovieClip;

      public function ContactsAPI(target:MovieClip){

            mc = target;

            mc.createTextField("statusTxt", 1, 25, 25, Stage.width, Stage.height);
            statusTxt = new TextField();
            statusTxt = mc.statusTxt;
            statusTxt.text = " Text: " + newline;

            var contact1:ContactEntry = new ContactEntry();
            contact1.setFirstName("Joseph");

            var contact2:ContactEntry = new ContactEntry();
            contact2.setFirstName("Claire");

            var contact3:ContactEntry = new ContactEntry();
            contact3.setFirstName("Andrew");

            var contact4:ContactEntry = new ContactEntry();
            contact4.setFirstName("Wan Ping");

            Contacts.addContact(this, Delegate.create(this, onAddContact), contact1);
            Contacts.addContact(this, Delegate.create(this, onAddContact), contact2);
```

continues

www.it-ebooks.info

LISTING 12-17 *(continued)*

```
            Contacts.addContact(this, Delegate.create(this, onAddContact), contact3);
            Contacts.addContact(this, Delegate.create(this, onAddContact), contact4);

      }

      private function onAddContact(owner:Object, status:Boolean, id:String):Void {

            if (status == true){
                  statusTxt.text += "Contact Successfully added. Id = ";
                  statusTxt.text += id + newline;
            }
      }
}
```

Filtering Contacts

The `ContactFilter` is another data type in the API and serves as the selection criteria parameter for retrieving contact entries through the `getEntries()` method. The `ContactFilter` also needs to be imported into the `ContactsAPI` before it can be used.

In Listing 12-18 you'll see the public `search()` method for the class, which allows you to pass in the name for the contact you want to filter by. Here, the `filterObj` filter object has its `fieldsToSearch` and `matchingValue` properties defined. These, incidentally, are the only properties that `ContactFilter` instances have. The object is passed as a parameter to `getEntries()` along with a reference to a callback method called `onGetEntries()` and two other parameters `startIndex` and `maxResults`, which are set to 0 and 2, respectively. This means the maximum number of entries returned will be 2, and the filtering will begin from the first contact in the list, which is represented by the `startIndex`.



LISTING 12-18: Using a ContactFilter object to filter contacts by first name or last name

```
public function search(contactName:String):Void{

      var startIndex:Number = 0;
      var maxResults:Number = 2;

      var filterObj:ContactFilter = new ContactFilter();
      filterObj.setFieldsToSearch("firstName|lastName");
      filterObj.setMatchingValue(contactName);

      Contacts.getEntries(this, Delegate.create(this, onGetEntries), filterObj,
startIndex, maxResults);

}
```

In Listing 12-19 you'll see the `onGetEntries()` function. When `getEntries()` is successful, the function will loop through the `entries` parameter invoking the `getFirstName()` call

www.it-ebooks.info

on each of the `ContactEntry` instances and setting the returned value to the `statusTxt` text field.



LISTING 12-19: The onGetEntries method for the contacts example

```
private function onGetEntries(owner:Object, status:Boolean, entries:Array):Void {

      if(status == true){
            for(var i:Number = 0; i < entries.length; i++){
                  statusTxt.text += entries[i].getFirstName() + newline;
            }
      }
}
```

Depending on what value is supplied to the `search()` method, the `getEntries()` call looks at both the `firstName` and `lastName` properties, because this was specified in the filter object.

The completed class will add several contacts to a device's phone book when it is instantiated. You can also use the class to search for contacts in the phone book. The following code snippet shows how to run the completed `ContactsAPI` example from the timeline and how to call the `search()` method.




```
import com.wrox.chapter12.ContactsAPI;

var contacts:FileDirectories = new ContactsAPI(this);

contacts.search("Wan Ping");
```

code snippet chapter12_Contacts.fla

When you create a test environment and run the example in Device Central, you'll see the test data presented. In the next section you'll take a look at using the File Service API.

USING FILE

The File Service API has methods you can use to retrieve information about the device's file system. You can use the API to read or write text-based files, preexisting files and create new files. You can also list or create new directories on the file System.

API Features

The `File` class has 14 methods:

- `addFileSystemChangedListener()` — Used to register a listener that is notified when a new file system is either added or removed from the root of the device
- `createDirectory()` — Used to create a new directory

www.it-ebooks.info

- `createFile()` — Used to create a new file
- `getFileInfo()` — Used to obtain the properties of a file or directory
- `listFiles()` — Used to obtain a filtered list of files and directories contained in a directory
- `listFileSystems()` — The currently mounted root file systems on a device
- `setHidden()` — Used to set the hidden attribute of the selected file to the value provided
- `setWritable()` — Used to set the selected file or directory writable attribute to the indicated value
- `setReadable()` — Used to set the file or directory readable attribute to the indicated value
- `writeFile()` — Used to write content to a file
- `readFile()` — Used to obtain the contents of a text file being pointed to
- `remove()` — Used to move a file or directory
- `removeFileSystemChangedListener()` — Used to unregister the listener associated with the file system change
- `rename()` — Used to rename a defined file or directory

There are also three data types associated with the API: `FileInfo`, `FileSystemData`, and `RootChangedEvent`. Each event handler function that returns information about a file or directory returns a `FileInfo` object in the parameters, including properties such as the modification date. If you need to, you can also set the attributes of a path to be `readable`, `writable`, or `hidden`. Renaming files is also an option. In addition, each `FileInfo` object returns a name path and URL, along with a size.

Several attributes of the `FileInfo` object can be obtained as well; these include:

- `isDirectory` — Indicates whether or not the object is a directory
- `isHidden` — Indicates whether or not the file or directory is hidden
- `isReadable` — Indicates whether or not the directory is readable
- `isWritable` — Indicates whether or not the file or directory is writable

All of these properties are private, so to gain access to them, you need to use the getter and setter methods of each property.

The `FileSystemData` class contains data about a file system, and includes properties such as `totalSize`, `availableSize`, `usedSize`, and the `path` to the file system. The `RootChangedEvent` class contains information about changed file systems, and stores properties for the `state` change, and the `rootName`.

Here, you'll cover the `listFiles()` method of the File API, including a demonstration of how to list files in various directories of the device, just as in a file explorer. The `chapter12_File.fla` accompanying this chapter contains the code on the first frame in the timeline that initializes the service.

www.it-ebooks.info

Reading Directories

In this section you'll follow the example to read and display the directories for a device, using the File Service API.

Mobile devices usually have one or more of the following file systems listed:

- `/c://` — Local drive or phone memory
- `/e://` — External drive or mass memory
- `/f://` — Secondary external drive or memory card

In Listing 12-20 you can see that the `File` class is imported into the example class for this section; it's called `FileAPI`. Here, a few variables are defined. The variable `folderPaths` is an array that will store the folder paths or files retrieved, while the `directoryPath` variable is set to the local device drive and will reference the currently selected path. In the constructor you'll see that a list and soft-keys component are attached to the target MovieClip container. The stub for the `onSoftKeyDown()` event handler is outlined here also.



LISTING 12-20: Creating the File API example

```
import com.sonyericsson.capuchin.file.File;
import mx.utils.Delegate;

class com.wrox.chapter12.FileAPI {

      private var folderPaths:Array;
      private var directoryPath:String = "/c:/";
      private var fileList:MovieClip
      private var softKeys:MovieClip;
      private var mc:MovieClip;

      public function FileAPI(target:MovieClip) {
            mc = target;
            fileList = mc.attachMovie("ListSingleRow", "fileList", 0);
            fileList._y = 30;
            fileList.addItems(["/c:/"]);

            softKeys = mc.attachMovie("SoftKeys", "softKeys", 1);
            softKeys._LSK = " /c:/";
            softKeys._RSK = " /e:/";
            softKeys._MSK = "list files";
            softKeys.onSoftKeyDown = Delegate.create(this, onSoftKeyDown);
      }

      private function onSoftKeyDown(key:String):Void {
            switch(key){
                  case "LSK":
                  break;
                  case "MSK":
                  break;
```

continues

www.it-ebooks.info

LISTING 12-20 *(continued)*

```
                case "RSK":
                break;
            }
        }
}
```

Two functions for the class are used: `listFiles()` and `onListFiles()`. The `listFiles()` method invokes the static `listFiles()` method of the File API, while the corresponding `onListFiles()` callback method simply sets each of the file paths in the specified directory, which is retrieved from the `filePaths` array parameter (see Listing 12-21). The left, right, and middle soft keys are also defined in `onSoftKeyDown()` so that the `listFiles()` method is called. When you run the example pressing the left soft key will set the directory path to "/c:/", while setting the right soft key will set the directory path to "/e:/". Pressing the middle soft key will retrieve the directory highlighted in the list component.


Available for download on Wrox.com

LISTING 12-21: Calling listFiles method in the File example

```
private function onSoftKeyDown(key:String):Void {
      switch(key){
            case "LSK":
                  directoryPath = "/c:/";
                  listFiles();
            break;
            case "MSK":
                  directoryPath = folderPaths[fileList.getFocusIndex()];
                  listFiles();
            break;
            case "RSK":
                  directoryPath = "/e:/";
                  listFiles();
            break;
      }
}

private function listFiles():Void {
      File.listFiles(this, Delegate.create(this, onListFiles), directoryPath, null);
}

private function onListFiles(owner:Object, status:Boolean, filePath:Array):Void {
      if(status == true){
            folderPaths = [];

            for (var i:Number=0; i <filePath.length; i++) {
                  folderPaths.push(filePath[i]);
            }

            fileList.addItems(folderPaths);
      }
}
```

www.it-ebooks.info

The code to run the example is shown in the following snippet, which must be placed on the first frame of the timeline:




```
import com.wrox.chapter12.FileAPI;

var file:FileAPI = new FileAPI(this);
```

code snippet chapter12_File.fla

After you create a test environment and run the example in Device Central, you'll see the test data presented. In the next section you'll take a look at using the I18n Service API.

USING I18N

The I18n Service API allows you to internationalize Project Capuchin applications through several static methods specially designed to help format the following:

- Currency
- Date and time
- Messages
- Numbers
- Percentages

The API essentially provides you with a smart way of handling differences between regions around the world. One obvious example is that in the UK dates are generally written as "dd/mm/yyyy," whereas in the US dates are written as "mm/dd/yyyy." So for a mobile application that presents the date for the FA Cup final as 06/05/2010, is this for 6th May or 5th June? This may be a subtle difference, but it highlights that locale has a bearing on how your application is determined. I18n should be considered as a significant aspect of development for global Flash mobile content.

API Features

Using I18n in your Flash mobile content will add value to your applications, enabling them to cater for multiple locations and regional settings. The API has the following methods:

- `compareStrings()` — Used to compare two strings using rules specific to set locale and given collation type
- `formatCurrency()` — Formats a currency amount using the locale-specific rules
- `formatDateTime()` — Formats a date or time instance using locale-specific rules
- `formatMessage()` — Formats a text with a variable number of ordered parameters
- `formatNumber()` — Formats a number
- `formatPercentage()` — Formats a percentage with the specified number of decimals using locale-specific rules

www.it-ebooks.info

- `getComparingLocales()` — Used to get a list of locales supported for string comparison using the `compareStrings()` and `sortStrings()` comparative methods
- `getFormattingLocales()` — Used to get a list of locales supported for formatting using the `formatCurrency()`, `formatDateTime()`, `formatMessage()`, `formateNumber()`, and `formatPercentage()` formatting methods
- `getLocale()` — Used to get the set locale of the device
- `setLocale()` — Used to set the locale for formatting, string comparison, and getting resources
- `sortStrings()` — Used to sort strings using rules specific to set locale and given collation type

For I18n we simply demonstrate the usage of `formatDateTime()`, `getLocale()`, `setLocal()`, and `formatCurrency()`.

> **I18N**
>
> I18n is an abbreviation derived from the word "Internationalization," 18 represents the number of letters between the "I" and the "n."

Internationalizing Currency

In each of the I18n examples, you will use the locale property, with either `getLocale()` or `setLocale()`. Both are static methods and require the `I18n` class to be imported.

The I18n API uses the three-letter code established by the International Organization for Standardization (ISO, `www.iso.org`) ISO 4217, as a reference to the names of currencies recognized around the world. A few three-letter codes are listed below and will be used in the first example:

- GBP — Pound sterling
- USD — U.S. dollar
- JMD — Jamaican dollar
- EUR — Euro
- MYR — Malaysian ringgit

You can either retrieve the locale of the device through the API or set one to cater for different locales. Using the static `formatCurrency()` method of the API, you can ensure that monetary values are written according to their specific regional format.

In Listing 12-22 you'll see several variables defined for the example. `currency` will hold reference to the selected currency, the `locale` is a reference to the locale, `pocketMoney` is a monetary value, and `formattedStr` will hold a reference to the formatted string. Then, there are the five currency objects defined with their three-letter code. Each of these has an `id`, `name`, `code`, and `symbol` property. The default currency is `gbp`, and this is assigned to `currency`.

www.it-ebooks.info



LISTING 12-22: Defining regional currency objects

```
import com.sonyericsson.capuchin.i18n.I18n;

var currency:Object;
var locale:String;
var pocketMoney:String = "1000000.00";
var formattedStr:String;

var gbp:Object = { id:0, name:"Pound Sterling", code:"GBP", symbol:"£" };
var usd:Object = { id:1, name:"US Dollar", code:"USD", symbol:"$" };
var jmd:Object = { id:2, name:"Jamaican Dollar", code:"JMD", symbol:"J$" };
var eur:Object = { id:3, name:"Euro", code:"EUR", symbol:"€" };
var myr:Object = { id:4, name:"Malaysian ringgit", code:"MYR", symbol:"RM" };

currency = gbp;
I18n.getLocale(this, onGetLocale);
```

The `onGetLocale()` callback method here captures the `status` of the API method and has the `localeStr` string parameter returned. It then sets this to the `locale` variable, which is the variable for a text field on the stage of the example `.fla` file. The method calls `formatCurrency()`, passing the code property of the `currency` object as parameter, which should be set to "GBP" at this stage. The value to be formatted, in this case `pocketMoney`, is also passed to call (see Listing 12-23).



LISTING 12-23: Formatting currency

```
function onGetLocale(owner:Object, status:Boolean, localeStr:String):Void {
      if(status == true){
            I18n.formatCurrency(this, formatHandler, currency.code, pocketMoney);
            locale = localeStr;
      }
}
```

In Listing 12-24 you can see that the callback for the `formatCurrency()` method handler has the `formattedValue` parameter returned. This is assigned to the `formattedStr` in the example, and because the currency code is set to "GBP," the currency symbol is placed in front of the `formattedValue`. The other regions could be captured to ensure that their locale is handled appropriately. This could be done in either the `onGetLocale()` or the format methods.



LISTING 12-24: The formatHandler method for the I18n example

```
function formatHandler(owner:Object, status:Boolean, formattedValue:String):Void {
      if(status == true) {
            switch(currency.code) {
```

continues

www.it-ebooks.info

LISTING 12-24 *(continued)*

```
                    case gbp.code:
                         formattedStr = currency.symbol + formattedValue;
                    break;
               }
          }
     }
```

The locale parameter used for setting the locale uses a pair of letter codes defined by ISO-639 and ISO-3166, for example, en-GB for United Kingdom, en-US for United States, or fr-FR for France. Each code is distinct for its locale and is defined by the language "en" and country "GB," for example. In addition, you can use the locale and format the date for a specified region.

In Listing 12-25 you'll see examples of regional objects for the United States, United Kingdom, France, Japan, and Sweden, with their `id`, `language`, `country`, and `code` properties defined, as well as an example of setting the locale using the `code` property (see Listing 12-25).



Available for download on Wrox.com

LISTING 12-25: Setting the locale using the setLocale method

```
import com.sonyericsson.capuchin.i18n.I18n;

var enUS:Object = { id:0,
                    language:"English",
                    country:"United States",
                    code:"en-US" };

var enGB:Object = { id:1,
                    language:"English",
                    country:"United Kingdom",
                    code:"en-GB" };

var frFR:Object = { id:2,
                    language:"French",
                    country:"France",
                    code:"fr-FR" };

var jpJP:Object = { id:3,
                    language:"Japanese",
                    country:"Japan",
                    code:"jp-JP" };

var svSE:Object = { id:4,
                    language:"Swedish",
                    country:"Sweden",
                    code:"sv-SE" };

I18n.setLocale(enGB.code);
```

www.it-ebooks.info

Formatting the Date and Time

Using I18n you can also format the date and time according to locale. Because the API uses primitive data types instead of the `Date` object, it makes it a little easier to know what's going on, and you don't have to worry about things such as time zone offsets.

The `formatDateTime()` method can be used to style dates based on one of the following style settings:

- `SHORT_DATE` — 10/25/09
- `LONG_DATE` — October 25, 2009
- `SHORT_TIME` — 12:19 PM
- `LONG_TIME` — 12:19:31 PM Z
- `SHORT_DATETIME` — 10/24/08 12:19 PM
- `LONG_DATETIME` — October 25, 2009 12:19:31 PM

When formatting dates, you need to import the `DateTime` data type class in addition to I18n. In Listing 12-26 you'll see a `DateTime` object defined, called `dateTime`, specifying all the date and time parameters. In this example we're setting date and time for May 25, 1977. Here, the `formatDateTime()` method is used with the `dateTime` and `style` parameter, which is set to display the `LONG_DATE` format.



LISTING 12-26: Formatting date and time using the formatDateTime method

```
import com.sonyericsson.capuchin.i18n.I18n;
import com.sonyericsson.capuchin.i18n.datatypes.DateTime;

var dateTime:DateTime = new DateTime();
dateTime.setYear(1977);
dateTime.setMonth(5);
dateTime.setDay(25);
dateTime.setHour(20);
dateTime.setMinute(30);
dateTime.setSecond(0);
dateTime.setMilisecond(0);

var formattedStr:String:
var style:String = "LONG_DATE";

I18n.formatDateTime(this, formatHandler, dateTime, style);

function formatHandler(owner:Object, status:Boolean, formattedValue:String) {
      formattedStr = formattedValue;
}
```

While you have not covered all the aspects of the I18n API, you can gather that it is relatively simple to come to grips with once you understand how the methods use the locale information. In the next section, you'll take a look at using the Messaging API.

www.it-ebooks.info

USING MESSAGING

Using the Project Capuchin Messaging API provides you with another way to send messages from your Flash Lite applications to other devices.

API Features

There are numerous methods available for the service, which include:

- `addBroadcastMessageReceivedListener()` — Listen for incoming messages from the operator
- `addMMSReceivedListener()` — Listen for incoming MMS messages
- `addSMSReceivedListener()` — Listen for incoming SMS messages
- `removeBroadcastMessageReceivedListener()` — Stop listening for incoming messages from the operator
- `removeMMSReceivedListener()` — Stop listening for incoming MMS messages
- `removeSMSReceivedListener()` — Stop listening for incoming SMS messages
- `sendMMS()` — Send text messages with attachments
- `sendSMS()` — Send text messages

In this section you'll simply take a look at sending SMS via the `sendSMS()` method.

Sending SMS

To use the API methods, you need to ensure that you have imported the `Messaging` class, and to send an SMS you need to specify the target mobile device's phone number, `mobileNo`, and the `port` number of the device and message, then call `Messaging.sendSMS()` (see Listing 12-27).



LISTING 12-27: Using the sendMessage method of Messaging API

```
import com.sonyericsson.capuchin.messaging.Messaging;
import mx.utils.Delegate;

class com.wrox.chapter12.MessagingAPI {

      private var mc:MovieClip;
      private var statusTxt:TextField;

      public var mobileNo:String = "+440123456789";
      public var message:String = "Message to Earl";
      public var port:Number = 90210;

      public function MessagingAPI(target:MovieClip) {

            mc = target;

            mc.createTextField("statusTxt", 1, 25, 25, Stage.width, Stage.height);
```

www.it-ebooks.info

```
            statusTxt = new TextField();
            statusTxt = mc.statusTxt;
            statusTxt.text = "Text: " + newline;
      }

      public function init():Void {
            Messaging.sendSMS(this, Delegate.create(this, onSend), mobileNo, port,
message);\
      }

      private function onSend(owner:Object, status:Boolean) {
            if (status == true) {
                  statusTxt.text = "Your message was successfully sent.";
            } else {
                  statusTxt.text = "Your message wasn't sent.";
            }
      }
}
```

Project Capuchin Messaging Service API requires an open port number on the device to be specified when using the `sendMessage()` method. It's recommended that port numbers above 50000 be used to avoid any interference with other applications on the device.

The Messaging API is quite similar to the S60 Services Platform Messaging API in terms of functionality and is not covered at length here.




```
import com.wrox.chapter12.MessagingAPI;

var messaging:MessagingAPI = new MessagingAPI(this);

messaging.init();
```

code snippet chapter12_Messaging.fla

When you create a test environment and run the example in Device Central you'll see the test data presented. In the next section you'll take a look at using the Persistency Service API.

USING PERSISTENCY

The Persistency API provides you with another way to store data on the device, similarly to `SharedObject` covered in Chapters 8 and 9.

API Features

A perfect example for using the Persistency service is a login form, like the one in the Twitter client application created in Chapter 9, where you used the shared objects to save a reference to a username.

www.it-ebooks.info

The API has three methods:

- `init()` — Used to initialize the Persistency Service
- `store()` — Used to save data
- `retrieve()` — Used to retrieve data

You'll cover each of the methods for the API.

Saving and Retrieving Data

The Persistency Service has a slightly different method to save and retrieve data than the methods of the `SharedObject` class. So here you will take a look at how the service uses the key name and storage name properties.

Defining Key and Storage Names

The key name is a reference to a value you want to save or retrieve. The "key" is simply a string that uniquely identifies a value in a particular storage location, while the "storage" can be thought of as a unique container on the device, holding a number of "key-value" references. Storage locations are referenced through the string representation of storage name.

Both the key and storage location name need to be referenced when any of the Persistency Services is called.

Login Example

The following snippets will guide you through an example of the Persistency API, which can be found in the `PersistencyAPI` class. The `chapter12_Persistency.fla` file contains the code that runs the example. The example is based on a login and uses a key called `userName` and the storage name `userStorage`.

To use the Persistency API, you need to ensure that you have imported the `Persistency` class, and then initialize the service using the `init()` method.

1. Start off with the code in Listing 12-28, which initializes the Persistency API and draws a button text field to view. The button will be used as a save button, called `saveButton`, and the text field, for holding the username, is called `usernameTxt`. In the parameters supplied to the `init()` method, you pass this as target reference call for the service, the callback handler `onInit()`, and the name of the storage location you want to access, which is set to `userStorage`. The stubs for `onInit()` and `onSave()` are also included here; you'll return to these shortly.



LISTING 12-28: Creating the Persistency API example

```
import com.sonyericsson.capuchin.persistency.Persistency;
import mx.utils.Delegate;

class com.wrox.chapter12.PersistencyAPI {
```

www.it-ebooks.info

```
        private var usernameTxt:TextField;
        private var saveButton:MovieClip;
        private var mc:MovieClip;

        public function PersistencyAPI(target:MovieClip){

              mc = target;
              Persistency.init(this, Delegate.create(this, onInit), "userStorage");
              draw();

        }

        public function draw():Void {
              saveButton = mc.attachMovie("PushButton", "saveButton", 1);
              saveButton._x = 10;
              saveButton._y = 100;
              saveButton.onPress = Delegate.create(this, onSave);

              mc.createTextField("usernameTxt", 2, 25, 25, 180, 30);

              var textFormat:TextFormat = new TextFormat();
              textFormat.font = "_sans";
              textFormat.leftMargin = 2;
              textFormat.size = 11;
              textFormat.color = 0x7EBAC0;

              usernameTxt.multiline = false;
              usernameTxt.border = true;
              usernameTxt.borderColor = 0xD5EAF4;
              usernameTxt.background = true;
              usernameTxt.backgroundColor = 0xCCCFEF;
              usernameTxt.type = "input";
              usernameTxt.setNewTextFormat(textFormat);
        }

        private function onInit():Void { }

        private function onSave():Void { }

}
```

2. Before completing `onInit()`, create two functions for the class called `getUsername()` and `onRetrieve()`. The `getUsername()` function should invoke the static `retrieve()` method of the Persistency Service, while `onRetrieve()` will be the callback handler. The `retrieve()` method requires both the key and storage names to be supplied as parameters (see Listing 12-29).



LISTING 12-29: The getUsername and onRetrieve methods for Persistency API example

```
public function getUsername():Void {
      Persistency.retrieve(this, Delegate.create(this, onRetrieve), "userName",
"userStorage");
}
```

continues

www.it-ebooks.info

LISTING 12-29 *(continued)*

```
private function onRetrieve(owner:Object, status:Boolean, value:String):Void {
      if(status == true) {
            usernameTxt.text = value;
      } else {
            trace("Unable to initiate persistencyObj.");
      }
}
```

3. Next, complete the `onInit()` method such that the `getUsername()` method is called if the service is available (see Listing 12-30).



LISTING 12-30: The onInit method for the Persistency API

```
private function onInit(owner:Object, status:Boolean):Void {
      if(status == true) {
            getUsername();
      } else {
            usernameTxt.text = "Unable to initialize Persistency API";
      }
}
```

4. Before completing `onSave()`, define the two methods to handle saving the username, called `saveUsername()` and `onSaveUsername()`. In `saveUsername()` you need to call the `store()` method of the Persistency API, to save the username retrieved from the `usernameTxt` text field. The `username` value needs to be a parameter for `saveUsername()`, and needs to be supplied to the `store()` method along with the reference to the key and the storage name and also the callback handler for the method called `onSaveUsername()` (see Listing 12-31).



LISTING 12-31: The saveUsername and onStore methods

```
public function saveUsername(username:String):Void {
      Persistency.store(this, Delegate.create(this, onSaveUsername), "userName",
"userStorage", username);
}

private function onSaveUsername(owner:Object, status:Boolean):Void {
      if(status == true) {
            trace("The username is saved.");
      } else {
            trace("unable to save");
      }
}
```

www.it-ebooks.info

5. Next, complete the event handler for the save button. This needs to call the `saveUsername()` method, passing it a reference to the value in `usernameTxt` (see Listing 12-32).



LISTING 12-32: The onSave method for the Persistency API

```
private function onSave():Void {
      saveUsername(usernameTxt.text);
}
```

In this example you've examined how to use the Persistency API to save data to and retrieve data from the device. The following code snippet shows how to run the completed `PersistencyAPI` example from the timeline:




```
import com.wrox.chapter12.PersistencyAPI;

var persistency:PersistencyAPI = new PersistencyAPI(this);
```

code snippet chapter12_Persistency.fla

When you create a test environment and run the example in Device Central, you'll see the test data presented. In the next section you'll take a look at using the Radio Service API.

USING RADIO

The Radio API is another cool feature of the Project Capuchin Service APIs that you can use to include Radio functionality for the device in your Flash Lite application. There are also methods to search for a radio station's broadcast signal frequency on either FM or AM modulations.

API Features

The API has numerous methods associated with it:

- `play()` — Used to play the radio
- `stop()` — Used to stop playing the radio
- `setFrequency()` — Used to tune to the given frequency or to the closest supported frequency
- `getFrequency()` — Used to get the frequency that the tuner has been tuned to
- `getModulation()` — Used to get the modulation in use
- `getMinFreq()` — Used to get the minimum frequency supported by the tuner with the given modulation
- `getMaxFreq()` — Used to get the maximum frequency supported by the tuner with the given modulation

www.it-ebooks.info

- `setSoundMode()` — Used to set the sound mode (stereo, mono or auto)
- `getSoundMode()` — Used to retrieve the sound mode in use (stereo, mono, auto)
- `listPresets()` — Used to get a list of presets
- `setPresetName()` — Used to set the preset name
- `setPreset()` — Used to configure the preset or creates a new one using given settings
- `usePreset()` — Used to tune the tuner with the settings specified in the preset
- `getNumberOfPresets()` — Gets the number of presets
- `searchFrequency()` — Used to search for the next broadcast signal
- `getRDSData()` — Used to get the current RDS Data
- `isRDSSignal()` — Returns the status of the RDS reception
- `isPlaying()` — Used to check if the radio is currently playing
- `setVolumeLevel()` — Used to set the volume level with values between 0 and 100
- `getVolumeLevel()` — Used to get the current volume level set
- `setMute()` — Used to mute or unmute the radio signal
- `isMuted()` — Used to get the mute state of the radio signal
- `getPreset()` — Used to get a specific preset
- `addRDSDataArrivedListener()` — Used to register a listener for RDS data
- `removeRDSDataArrivedListener()` — Used to remove the RDS data listener

For this API we demonstrate the usage of `getFrequency()`, `getModulation()`, `getSoundMode()`, `listPresets`, `start()`, `addRDSDataArrivedListener()`, `usePreset()`, `searchFrequency()`, and `stop()`. The `chapter12_Radio.fla` example contains the call for the example class created for the API, which is called `RadioAPI`.

Listening to the Radio

In Listing 12-33 you'll see the beginning of the `RadioAPI` class, where the properties and the constructor for the `RadioAPI` class are defined. The completed class will allow you to connect to a device's radio, and the example uses several text fields to display information about the radio station, such as the frequency, modulation, RDS information, and sound mode. In the constructor, each asset is assigned to the stage; there is a list component to hold presets, and a soft keys component for initializing the Radio. There is also `Key` listener to capture the left soft key. To use the Radio API, the `Radio` class from the API needs to be imported, along with its related data types. For class-based development, the `Delegate` also needs to be imported.

www.it-ebooks.info



LISTING 12-33: Creating the Radio API example

```
import com.sonyericsson.capuchin.radio.Radio;
import com.sonyericsson.capuchin.radio.datatypes.RDSData;
import com.sonyericsson.capuchin.radio.datatypes.Preset;
import mx.utils.Delegate;

class com.wrox.chapter12.RadioAPI {
      private var mc:MovieClip;
      private var radioStarted:Boolean;
      private var presetList:MovieClip;
      private var rdsTxt:TextField;
      private var freqTxt:TextField;
      private var soundTxt:TextField;
      private var modTxt:TextField;
      private var softKeys:MovieClip;

      public function RadioAPI(target:MovieClip) {

            mc = target;

            var txtFormat:TextFormat = new TextFormat();
            txtFormat.size = 48;

            var depth1:Number = mc.getNextHighestDepth();
            mc.createTextField("rdsTxt", depth1, 5, 0, 240, 40);

            rdsTxt = new TextField();
            rdsTxt = mc.rdsTxt;
            rdsTxt.textColor = 0x000000;
            rdsTxt.text = "RDS";

            var depth2:Number = mc.getNextHighestDepth();
            mc.createTextField("freqTxt", depth2, 5, 22, 240, 80);

            freqTxt = new TextField();
            freqTxt = mc.freqTxt;
            freqTxt.textColor = 0x000000;
            freqTxt.text = "Frequency Values";
            freqTxt.setNewTextFormat(txtFormat);

            var depth3:Number = mc.getNextHighestDepth();
            mc.createTextField("soundTxt", depth3, 5, 80, 240, 20);

            soundTxt = new TextField();
            soundTxt = mc.soundTxt;
            soundTxt.textColor = 0x000000;
            soundTxt.text = "Sound Mode";

            var depth4:Number = mc.getNextHighestDepth();
            mc.createTextField("modTxt", depth4, 90, 80, 240, 20);
```

continues

www.it-ebooks.info

LISTING 12-33 *(continued)*

```
            modTxt = new TextField();
            modTxt = mc.modTxt;
            modTxt.textColor = 0x000000;
            modTxt.text = "Modulation";

            var depth5:Number = mc.getNextHighestDepth();
            presetList = mc.attachMovie("ListSingleRow", "presetList", depth5);
            presetList._y = 100;
            presetList._emptyListText = "Preset List";

            var depth6:Number = mc.getNextHighestDepth();
            softKeys = mc.attachMovie("SoftKeys", "softKeys", depth6);
            softKeys._LSK = "";
            softKeys._RSK = "";
            softKeys._MSK = "SEARCH";
            softKeys._isLSKEnabled = false;
            softKeys._isRSKEnabled = false;
            softKeys.onSoftKeyDown = Delegate.create(this, onSoftKeyDown);

            var keyListener:Object = new Object();
            keyListener.onKeyDown = Delegate.create(this, onKeyDownHandler);
            Key.addListener(keyListener);
        }
        public function onSoftKeyDown():Void {}

        public function onKeyDownHandler():Void {}

    }
```

To use the radio you need to ensure that it has actually been initialized. The `start()` method will return the status of the initialization, and this will require a callback. Once you know that the radio has been initialized, you can retrieve initial settings for it. The `onStart()` method, shown in Listing 12-34, retrieves the status and then calls a method named `initSettings()`.


Available for download on Wrox.com

LISTING 12-34: The onStart method for the Radio API example

```
public function onStart(owner:Object, status:Boolean):Void {
      radioStarted = status;
      initSettings();
}
```

The `initSettings()` function checks to see if the radio has started and then calls the `getFrequency()`, `getModulation()`, and `getSoundMode()` methods of the API to retrieve their respective values (see Listing 12-35). Each of the methods called has its own callback handler, which you'll take a look at next.

www.it-ebooks.info



LISTING 12-35: The initSettings method for the Radio API example

```
private function initSettings():Void {
      if(radioStarted){
            Radio.getFrequency(this, Delegate.create(this, onGetFrequency));
            Radio.getModulation(this, Delegate.create(this, onGetModulation));
            Radio.getSoundMode(this, Delegate.create(this, onGetSoundMode));
      }
}
```

The `onGetFrequency()` method is delegated to `getFrequency()`. In the parameters for the function, the frequency, `freq`, is used and assigned to the text field `freqTxt` (see Listing 12-36).



LISTING 12-36: Retrieving the radio frequency

```
private function onGetFrequency(owner:Object, status:Boolean, freq:Number):Void {
      freqTxt.text = String(freq / 10000) + " MHz";
}
```

The `onGetModulation()` method is delegated to `getModulation()`. In the parameters for the function, the frequency, `modType`, is used and is assigned to the text field `modTxt` (see Listing 12-37).



LISTING 12-37: Retrieving the modulation type

```
private function onGetModulation(owner:Object, status:Boolean, modType:String):Void
{
      modTxt.text = modType.toUpperCase();
}
```

The `onGetSoundMode()` method is delegated to `getSoundMode()`. In the parameters for the function, the sound mode, `mode`, is used and assigned to the text field `soundTxt` (see Listing 12-38).



LISTING 12-38: Retrieving the sound mode

```
private function onGetSoundMode(owner:Object, status:Boolean, mode:String):Void {
      soundTxt.text = mode;
}
```

In Listing 12-39 you'll see the `onSoftKeyDown()` function, which handles the key input. The middle key of the soft-key component is used to call the `start()` method of the API. It also sets the labels for the all the keys and then disables itself by setting the `_isMSKEnabled` property to `false`.

www.it-ebooks.info



LISTING 12-39: The onSoftKeyDown method for the Radio API example

```
private function onSoftKeyDown(key:String):Void {
      switch(key){
            case "LSK":
            break;
            case "MSK":
                  softKeys._LSK = "Tune – ";
                  softKeys._RSK = "Tune ++";
                  softKeys._MSK = "";

                  softKeys._isLSKEnabled = true;
                  softKeys._isRSKEnabled = true;
                  softKeys._isMSKEnabled = false;

                  Radio.start(this, Delegate.create(this, onStart));
                  Radio.addRDSDataArrivedListener(this, Delegate.create(this,
onRDSData));\
                  Radio.listPresets(this, Delegate.create(this, onListPresets));
            break;
            case "RSK":
            break;
      }
}
```

In addition to the `start()` method being called, the `addRDSDataArrivedListener()` and `listPresets()` methods of the API are invoked, with their callback handlers defined. The `onRDSData()` function will be discussed shortly; first, take a look at the `listPresets()` method.

Retrieving Presets

Presets are predefined frequencies that you can tune the radio to, instead of having to increment the frequency through a range.

In Listing 12-40 you'll see the `onListPresets()` function. In the parameters for the function, a list of `presets` is obtained. Each name for the preset is retrieved via the `getName()` method of the `Preset` data type. The names are then assigned to the list component `presetList`. The `onSelectPreset()` method shown invokes another call to the `usePreset()` method of the API, which simply requires the number of the preset selected. You can see that the focus index is retrieved from the `presetList` and used for this purpose. The `onUsePreset()` function then calls `initSettings()`. The idea here is to restart the radio with the new preset.



LISTING 12-40: Using the usePreset to retrieve presets

```
private function onListPresets(owner:Object, status:Boolean, presets:Array):Void {
      if (status != false) {
            var i:Number;
            var nameList:Array = [];
            var position:String;
```

www.it-ebooks.info

```
            for (i = 0; I < presets.length; i++) {
                  position = (i+1).toString();
                  nameList.push(position + ". " + presets[i].getName());
            }

            presetList.addItems(nameList);
            presetList.onSelect = Delegate.create(this, onSelectPreset);
      }
}

private function onSelectPreset():Void {
      var preset:Number = presetList.getFocusIndex() + 1;
      Radio.usePreset(this, onUsePreset, preset);
}

private function onUsePreset():Void {
     initSettings();
}
```

Using RDS Information

If you remember, `onRDSData()` is the callback for the `addRDSDataArrivedListener()` method. RDS (Radio Data System) can be used for station identification and, thus, the API provides the `RDSData` data type class for accessing properties such as the name of the radio service property, `programmeService`, and the radio text message property, `radioText`, which is transmitted from the station over RDS. Both these values are retrieved via the getter and setter methods `getProgrammeService()` and `getRadioText()` in `onRDSData()` (see Listing 12-41).



LISTING 12-41: Using the getProgrammeService and getRadioText methods to retrieving RDS data

```
public function onRDSData(owner:Object, status:Boolean, rds:RDSData) {
      rdsTxt.text = rds.getProgrammeService() + ", " + rds.getRadioText();
}
```

Searching for Radio Frequencies

Returning to the `onSoftKeyDown()` function, the `searchFrequency()` method of the API is called by pressing the left and right soft keys, to search for various frequencies. The `searchFrequency()` method uses a Boolean as an additional parameter, which conducts a scan for frequencies within the modulation's frequency range. Pressing the right soft key will set the frequency search forward, while the left soft key will set the frequency search in the opposite direction. If the scan reaches the end of the frequency range it starts from the beginning (see Listing 12-42).



LISTING 12-42: The onSoftKeyDown method for the Radio API example

```
private function onSoftKeyDown(key:String):Void {
      switch(key){
            case "LSK":
```

continues

www.it-ebooks.info

LISTING 12-42 *(continued)*

```
                Radio.searchFrequency(this, Delegate.create(this,
onSearchFrequency), false);
            break;
            case "MSK":
                softKeys._LSK = "Tune – ";
                softKeys._RSK = "Tune ++";
                softKeys._MSK = "";

                softKeys._isLSKEnabled = true;
                softKeys._isRSKEnabled = true;
                softKeys._isMSKEnabled = false;

                Radio.start(this, Delegate.create(this, onStart));
                Radio.addRDSDataArrivedListener(this, Delegate.create(this,
onRDSData));
                Radio.listPresets(this, Delegate.create(this, onListPresets));
            break;
            case "RSK":
                Radio.searchFrequency(this, Delegate.create(this,
onSearchFrequency), true);
            break;
        }
}
```

As with the `onGetFrequency()` function, `onSearchFrequency()` has the frequency as a parameter. The frequency values returned are measured in 100 Hz (see Listing 12-43).



LISTING 12-43: The onSoftKeyDown method for the Radio API example

```
public function onSearchFrequency(owner:Object, status:Boolean, freq:Number):Void {
        freqTxt.text = (freq / 10000).toString() + " MHz";
        initSettings();
}
```

Lastly, the `onKeyDownHandler()` method is used to stop the Radio, which invokes `fscommand2("Quit")` via the `onStop()` callback method (see Listing 12-44).



LISTING 12-44: The onKeyDownHandler method for the Radio API example

```
public function onKeyDownHandler():Void {
        switch (Key.getCode()) {
            case 48:
                Radio.stop(this, Delegate.create(this, onStop));
            break;
        }
}
```

www.it-ebooks.info

```
public function onStop(owner:Object, status:Boolean):Void {
      fscommand2("Quit");
}
```

The following code snippet shows how to run the completed `RadioAPI` example from the timeline:




```
import com.wrox.chapter12.RadioAPI;

var radio:RadioAPI = new RadioAPI(this);
```

code snippet chapter12_Radio.fla

When you create a test environment and run the example in Device Central, you'll see the test data presented. You can only use the radio on a device. Some devices require you to use an earphone accessory with the radio (as an aerial), so be aware of this.

OTHER PROJECT CAPUCHIN SERVICES

The Location and Multimedia API also have great features that you can include as part of your Flash Lite applications. Some of these features should sound familiar.

Location

The Location Service API has an array of methods for retrieving locations and managing landmark data for Project Capuchin–based devices, allowing you to provide the following functionality to your Flash Lite applications:

- Retrieving the device location
- Managing Landmarks

The API is comparable to the Location and the Landmark S60 Platform Services, all rolled into one!

Multimedia

The Multimedia API provides "out-of-the-box" functionality for processing audio files on Sony Ericsson devices through the Flash Lite player, including:

- Audio file playback
- Voice recording
- Controlling volume
- Playing the device MIDI instrument

You covered audio playback in Chapter 7 and many of the features discussed, such as play, pause, controlling volume, and obtaining the duration, can be used as part of the Multimedia API as a set of getter and setter methods, and in some instances provide a simpler alternative.

www.it-ebooks.info

For more information on these two, and all the APIs covered in this chapter, visit the Sony Ericsson Developers World website.

SUMMARY

In this chapter, you explored the Project Capuchin Platform Services API and learned new ways to extend Flash Lite standalone applications for Sony Ericsson–based devices. You also learned how to utilize test data in Device Central, and in the next chapter you'll learn how to package Project Capuchin-based applications using the Swf2Jar tool. This will allow you to run content on a Project Capuchin supported device.

For more information on Project Capuchin, visit `http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp`.

www.it-ebooks.info

13

Packaging Flash Lite Applications for Distribution

WHAT'S IN THIS CHAPTER?

- Utilizing Symbian SDK tools to create application installer files
- Using Swf2Jar to package content targeted for devices
- Signing your packages
- Distribution opportunities

Okay, so you are now at the stage where you have created your Flash mobile content, fully tested your application in Adobe Device Central, and tested it again and again on the plethora of handsets that you own. Now you're ready to take it to market. Where do you start?

In Chapters 5 through 11, you learned how to create content with resource files that are associated with the main application `swf` file. In this chapter, you'll explore how to package Flash Lite content that is ready for distribution to the consumer.

PACKAGING CONTENT

Learn to package your Flash Lite content, and you'll be able to overcome some of the obstacles in the way of distributing your Flash Lite content, including:

- Distributing applications with multiple resource files.
- Targeting the right platform.
- Installing files to optional paths.

www.it-ebooks.info

- Ensuring files are installed to their correct locations.
- Providing a single access point.

The content you develop may not always be contained in a single `.swf` file, and in fact, it is more likely that your mobile content will require multiple resources and data files associated with the main application. As with some of the content you've developed in this book, application resources such as `.xml` and image files that reside externally to the main `.swf` file would make it a complex affair for the end user to transfer content to his or her own device. So ensuring multiple resource files are installed in their correct locations is one of the benefits to packaging content.

The packaging process for Flash Lite applications is one that essentially creates a single installation file for your content. Once an application has been packaged, you can provide a single access point, in which all the resource files associated with the main `.swf` file are combined, ready to be installed through a file format recognized by a particular device.

The installation file should provide a seamless way for the consumer to install and run the Flash Lite content you've developed, the way you intended, and without any installation complications. The file will be different, depending on the platform you are targeting.

Packaging for Different Platforms

Relying on the end user to transfer your files to the correct folders on the device could potentially be a major headache, which most likely will result in your application not being run or installed properly at all.

Windows Mobile Platform

When creating content for installation on the Windows Mobile platform, you need to package your content in Cabinet (CAB, `.cab`) files.

Symbian S60 Platform

If you are targeting Nokia devices that use the S60 platform on Symbian OS, you need to package content as a Symbian Installation Source (SIS, `.sis` *or* `.sisx`) file. You can do this manually by using a command-line tool, which will be covered shortly. There are also online packaging tools that could accomplish the same task, such as the Nokia Flash Packaging Tool.

Java Platform

For targeting Sony Ericsson devices that use the Java Platform, and devices that support Project Capuchin, you can package content as Java Archive (JAR, `.jar`) files. To do this you can use the Swf2Jar tool. You'll cover this packaging method first.

USING SWF2JAR

In Chapter 12 you should have downloaded and installed the Project Capuchin Kit component. This extension includes a command plug-in which allows you to build and package your Flash Lite content using Swf2Jar directly from Flash CS4, a feature worth noting. The Java files essentially

www.it-ebooks.info

run as a MIDlet, which is a Java application framework for the Mobile Information Device Profile (MIDP) specification for running Java on embedded devices. After packaging your content, the `.jar` file can be transferred to a Sony Ericsson device that supports Project Capuchin. This is actually one of the easiest ways to package Flash Lite content for distribution. When you run the content on the phone, the packaged Flash content is automatically played as part of the MIDlet.



You can install Swf2Jar without having to install the Capuchin Kit MXP extension for Flash CS4. At the time of writing, the current release of Swf2Jar is version 2.2. This can be downloaded from the Sony Ericsson Developer Web site at `http://developer.sonyericsson.com/site/global/docstools/projectcapuchin/p_projectcapuchin.jsp`.

Swf2Jar 2.2 requires Java Runtime Environment (JRE) version 1.5 or higher, this can be downloaded from the Sun Web site at `http://java.sun.com/j2se`.

Creating a .jar File

The command to run Swf2Jar in Flash CS4 can be found in the Commands menu. Before running the command, you need to ensure that your `.fla` file is open. Here's a quick run through:

1. Open the `chapter13_swf2Jar.fla`. On the stage you'll see that there is a text field with the text "Swf2Jar Example".
2. Select Commands ➪ Create Capuchin Application, to run the command to package the file. You'll notice that the Publishing panel appears. This happens because the `.fla` file is being compiled into the SWF. A command-line window also appears shortly before launching the Swf2Jar application.
3. In the Swf2Jar application window that opens, take note of the three tabs: Required, Security, and Resources. In order to package your file, only a few fields need to be filled in on the Required tab; all the fields in the Security and Resources tabs are optional.

   The following information must be specified in the Required tab:

   - **Flash file —** This is where you set the main `.swf` file to be included in the MIDlet application. When the Required tab is shown, the Flash file field should be populated with the absolute path to your content, the output `.swf`. The file should be located in the same path as your `.fla` file. Alternatively, you can click the Folder icon to the right and browse to the Flash file you want to include.
   - **Application Name —** This is where you specify the name of the application. This value will be shown in the phone in the list of installed applications.
   - **Author —** This is where you can specify a name for the author of the application.

www.it-ebooks.info

- The following information in the Required tab is optional:
  - **Author URL** — This is where you can set a URL for the author.
  - **Application folder** — This is where you can set the directory where the application will be installed. You can select from either the Applications or Games folder on a Sony Ericsson device.
  - **Icon** — This is where you can specify an icon for the application, which is shown in the phone. You can provide an icon in the `.png` or `.gif` file format. To select the icon, click the folder icon to the right and browse to the icon file. It is also possible to drag and drop the image file into the icon area.

4. Ensure that the Flash file is a path to the "chapter13_swf2jar.swf", the Application Name is "Chapter 13 Swf2Jar", and the Author is set to "Wrox". When all required fields have been filled with information, notice that the Create button becomes enabled.
5. Select "Applications" as the default folder the application should be installed to.
6. Provide an icon for the application. An icon has been created for this step, called `icon.png`, which can be found in the `.zip` file accompanying this chapter. When using your own icons you need to ensure that they are 24x24 px.
7. Next press the Create button to begin packaging the `.swf` file into the `.jar` file.
8. Lastly, in the Choose destination folder dialog box that opens, select where you want the `.jar` file to be output to.

Figure 13-1 shows the Swf2Jar application, with all the required fields supplied.

Once you've selected your destination folder, your package will be created. If you followed the steps correctly, the file should be named "Chapter 13 Swf2Jar.jar". In the same folder you'll notice there is a Java Application Descriptor (JAD, `.jad`) file present, called "Chapter 13 Swf2Jar.jad". This file describes the contents of the `.jar` file. It is used by some mobile devices to describe what the application is, usually for installation purposes.

You can now transfer the `.jar` file to a compatible Sony Ericsson device with Project Capuchin applications supported.



FIGURE 13-1

www.it-ebooks.info



Not covered in detail here is the information in the Security tab, which is optional. Here, it is possible to set permissions and sign the application. The tab contains two groups: Permissions and Signature. The Permissions area specifies the permissions required for the application. You'll notice that there are two permissions that are predefined:

1. **Access Local Files** — *Adds permissions to read and write local files*
2. **Access Network** — *Provides support for HTTP and HTTPS connections*

Application Signing is broadly covered later in the chapter.

Next, you'll learn how to package `.sis` files using the command-line tool in Windows.

CREATING A .SIS FILE

For this section you will need to ensure that you have the Symbian S60 Platform SDK installed, which is available from the Forum Nokia Web site at `http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Platforms/S60_Platform_SDKs/`. Choose "Download all-in-one S60 SDKs" and then "3rd Edition, FP2 v1.1".

The process for creating a `.sis` file is highlighted in Figure 13-2, and you'll cover each of the processes shown here in detail throughout this chapter.

You will use the MakeSIS tool in the first part of the process for creating a `.sis` file. This can be located in the tools folder of your Symbian SDK installation path, for example: `<installation path>/Symbian/tools/makesis.exe`. In the same path, there are the MakeKeys and SignSIS tools, which are covered later.

But before you delve into creating installer files for your own applications, you should request a Unique Identifier (UID) reference.

Obtaining a Unique Identifier (UID)

The UID is a 32-bit number required by the Symbian OS to distinctively reference applications on the device. You need to be registered with Symbian Signed (`https://www.symbiansigned.com/`) for obtaining UIDs for your applications.

For the purpose of this section, you can create a Symbian 3rd Edition `.sis` file using the UID value `0xE0000001`. This value is from the Unprotected Range of Class E (0xE0000000– 0xEFFFFFFF), used for development purposes only. More information on UIDs can also be found at the Symbian Signed Web site.

www.it-ebooks.info

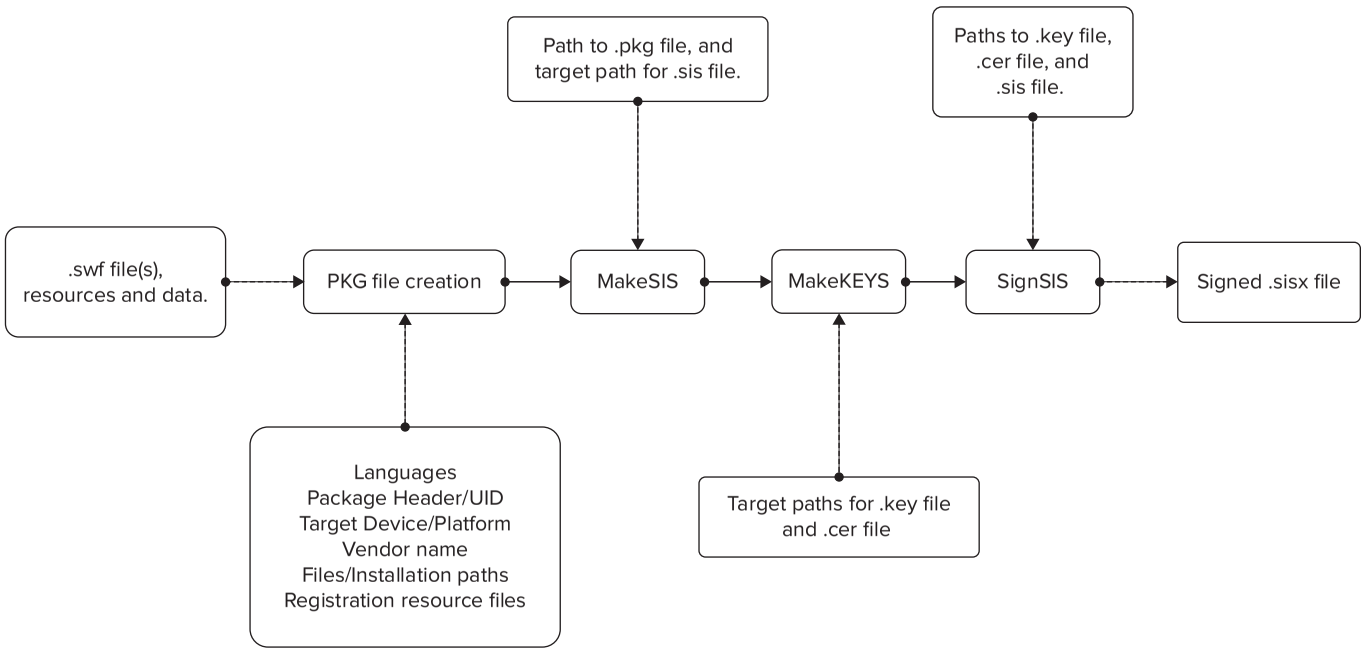


FIGURE 13-2

www.it-ebooks.info

Creating the Package (PKG) File

Before you use the command line to create your `.sis` file, you need to write a Symbian Package (PKG, `.pkg`) file. The `.pkg` file is essentially a list of instructions and properties that you need for the creation of a `.sis` file. This allows you to define the details of the installation of your application onto a device. The file is simply a text file, and so to create one all you need is a text editor.

The file contains all elements you need for producing your `.sis` file, including:

- Defining the application language
- SIS package header and name
- Product or platform version compatibility
- Vendor name
- File paths for the main SWF and associated resources

Listing 13-1 shows the completed `imageViewer.pkg` file, which defines the package and content for the Image Viewer application, created in Chapter 8. Take note of the lines that each directive falls on.



LISTING 13-1: The imageViewer.pkg file for the Chapter 8 Image Viewer application

```
&EN
#{"imageViewer"},(0xE0000001),1,0,0
[0x102752AE],0,0,0,{"S60ProductID"}
%{"JGAnderson"}
:"JGAnderson"
"C:\wrox\chapter13\imageViewer.swf"-"!:\Data\Others\Wrox\imageViewer.swf"
"C:\wrox\chapter13\images.xml"-"!:\Data\Others\Wrox\images.xml"
"C:\wrox\chapter13\images\01_lrg.png"-"!:\Data\Others\Wrox\images\01_lrg.png"
"C:\wrox\chapter13\images\01_sml.png"-"!:\Data\Others\Wrox\images\01_sml.png"
"C:\wrox\chapter13\images\02_lrg.png"-"!:\Data\Others\Wrox\images\02_lrg.png"
"C:\wrox\chapter13\images\02_sml.png"-"!:\Data\Others\Wrox\images\02_sml.png"
"C:\wrox\chapter13\images\03_lrg.png"-"!:\Data\Others\Wrox\images\03_lrg.png"
"C:\wrox\chapter13\images\03_sml.png"-"!:\Data\Others\Wrox\images\03_sml.png"
"C:\wrox\chapter13\images\04_lrg.png"-"!:\Data\Others\Wrox\images\04_lrg.png"
"C:\wrox\chapter13\images\04_sml.png"-"!:\Data\Others\Wrox\images\04_sml.png"
"C:\wrox\chapter13\images\05_lrg.png"-"!:\Data\Others\Wrox\images\05_lrg.png"
"C:\wrox\chapter13\images\05_sml.png"-"!:\Data\Others\Wrox\images\05_sml.png"
"C:\wrox\chapter13\images\06_lrg.png"-"!:\Data\Others\Wrox\images\06_lrg.png"
"C:\wrox\chapter13\images\06_sml.png"-"!:\Data\Others\Wrox\images\06_sml.png"
"C:\wrox\chapter13\imageViewer.rsc"-"!:\resources\apps\imageViewer.rsc"
"C:\wrox\chapter13\imageViewer.mif"-"!:\resources\apps\imageViewer.mif"
"C:\wrox\chapter13\imageViewer_reg.rsc"-"!:\private\10003a3f\import\apps\
imageViewer_reg.rsc"
```

Examining the imageViewer.pkg File Contents

In this section you'll take a look at the contents of the `imageViewer.pkg` file in more detail.

www.it-ebooks.info

Language

The first line of the `imageViewer.pkg` file starts with an ampersand `&`, followed by the languages supported by the application; the language is specified here simply as the UK English language code `EN` (see Listing 13-2).



LISTING 13-2: The defined Language for imageViewer.pkg

```
&EN
```

Table 13-1 lists some example languages that can be used in a `.pkg` file definition.

TABLE 13-1: Examples of Language Codes

LANGUAGE	CODE
US English	AM
UK English	EN
French	FR
German	GE
Thai	TH
Japanese	JP

Package Header

The second line, which starts with the hash symbol `#`, represents the `.sis` file package header and name. It should contain three values: the name of the application in curly brackets (here this is set as `{"imageViewer"}`); then the UID of the application in parenthesis, for example, `(0xE0000001)`; and finally a version number given to the application. This is represented by the major, minor, and build numbers of the application, defined here as `1,0,0`. Each value is separated from the next by a comma (see Listing 13-3).



LISTING 13-3: The defined Package Header for imageViewer.pkg

```
&EN
#{"imageViewer"},(0xE0000001),1,0,0
```

Target Device or Platform

The third line of the file represents the Symbian platform for the version you are targeting, and just as with the Package Header, there are three values to specify. For the first value, you have the choice of specifying a particular device's Product UID, or you can specify the more generic Platform UID,

www.it-ebooks.info

which is a value representing the version S60 platform. When specifying either Product UID or Platform UID, you are effectively defining which devices or platforms your application should be compatible with.

The example in Listing 13-4 targets the S60 3rd Edition FP2 platform, with the value `[0x102752AE]`. The next value represents the version number; this is set to `0,0,0`. The third value on this line is the tag for the S60 Platform, `{"S60ProductID"}`. Each value is separated from the next by a comma.



LISTING 13-4: The defined Target Platform for imageViewer.pkg

```
&EN
#{"imageViewer"},(0xE0000001),1,0,0
[0x102752AE],0,0,0,{"S60ProductID"}
```

Table 13-2 contains a list of Platform UIDs covering several editions of the S60 Platform.

TABLE 13-2: Symbian S60 Platform UIDs

PLATFORM NAME	PLATFORM UID
S60 5th Edition	0x1028315F
S60 3rd Edition, FP2	0x102752AE
S60 3rd Edition, FP1	0x102032BE
S60 3rd Edition	0x101F7961
S60 2nd Edition FP2	0x10200BAB

Table 13-3 contains a list of example Product UIDs for some popular Nokia devices.

TABLE 13-3: Nokia Devices with Corresponding Product UIDs and Platform UIDs

DEVICE NAME	PRODUCT UID	PLATFORM UID
Nokia N97	0x20014DDD	0x1028315F
Nokia E71	0x2000249b	0x102032BE
Nokia 5800	0x2000DA56	0x1028315F
Nokia N95 8GB	0x20002D84	0x102032BE
Nokia 6210	0x2000DA54	0x102752AE

www.it-ebooks.info



PLATFORM UID

There is a slight difference between the syntax for defining the Platform UID for S60 2nd Edition and for S60 3rd Edition onwards.

When targeting the S60 2nd Edition devices, you supply the Platform UID in rounded brackets:

```
(0x101F6F88), 0, 0, 0, {"Series60ProductID"}
```

Whereas for the S60 3rd Edition, you use a square bracket:

```
[0x102752AE], 0, 0, 0, {"S60ProductID"}
```

2nd Edition Symbian devices are unlikely to reappear, which is something to be aware of.

As another point of interest, the UID values for Windows Mobile consist of a 32-unit alphanumeric code separated at intervals by dashes: for example, 9DCABC1B-1B2F-4605-984D-B36FA1A26CA3. Whereas for Symbian S60 the value is considerably shorter, with a total of 10 characters: for example, 0xED8E364B.

From Table 13-3 you should notice the variation in values for a device model's product UID. You have to be very careful when you manually create a `.pkg` file, especially when you want to target the installation for a particular device. For instance, to target your installation to the Nokia N97, you would need to define the following line in a `.pkg` file: `[0x2000249b], 0, 0, 0, {"Nokia N97 ID"}`.

Vendor Name

The value preceded by the percentage symbol `%` is the "localized" version of the vendor, which in this case is defined by the value `{"JGAnderson"}`. The following line proceeded by the colon `:` is the unique vendor name; this value is set to `JGAnderson` (see Listing 13-5). The vendor's name is actually specified twice, on the following lines.



Available for download on Wrox.com

LISTING 13-5: The defined Vendor Name for imageViewer.pkg

```
&EN
#{"imageViewer"},(0xE0000001),1,0,0
[0x102752AE],0,0,0,{"S60ProductID"}
%{"JGAnderson"}
:"JGAnderson"
```

Files to Package and Installation Paths

Following the vendor declarations are lines representing each file that needs to be packaged into the `.sis` file, along with its corresponding installation location. There are 14 files in total, and

www.it-ebooks.info

each will need to be packaged into the `.sis` file for the Image Viewer. Each line has the location of the file; here `C:\wrox\chapter13\` is the base path to the Image Viewer directory, and essentially everything leading up to the hyphen (-) represents a file. Following the hyphen is the target installation directory for the device (see Listing 13-6).



LISTING 13-6: The files to package and installation paths for imageViewer.pkg

```
&EN
#{"imageViewer"},(0xE0000001),1,0,0
[0x102752AE],0,0,0,{"S60ProductID"}
%{"JGAnderson"}
:"JGAnderson"
"C:\wrox\chapter13\imageViewer.swf"-"!:\Data\Others\Wrox\imageViewer.swf"
"C:\wrox\chapter13\images.xml"-"!:\Data\Others\Wrox\images.xml"
"C:\wrox\chapter13\images\01_lrg.png"-"!:\Data\Others\Wrox\images\01_lrg.png"
"C:\wrox\chapter13\images\01_sml.png"-"!:\Data\Others\Wrox\images\01_sml.png"
"C:\wrox\chapter13\images\02_lrg.png"-"!:\Data\Others\Wrox\images\02_lrg.png"
"C:\wrox\chapter13\images\02_sml.png"-"!:\Data\Others\Wrox\images\02_sml.png"
"C:\wrox\chapter13\images\03_lrg.png"-"!:\Data\Others\Wrox\images\03_lrg.png"
"C:\wrox\chapter13\images\03_sml.png"-"!:\Data\Others\Wrox\images\03_sml.png"
"C:\wrox\chapter13\images\04_lrg.png"-"!:\Data\Others\Wrox\images\04_lrg.png"
"C:\wrox\chapter13\images\04_sml.png"-"!:\Data\Others\Wrox\images\04_sml.png"
"C:\wrox\chapter13\images\05_lrg.png"-"!:\Data\Others\Wrox\images\05_lrg.png"
"C:\wrox\chapter13\images\05_sml.png"-"!:\Data\Others\Wrox\images\05_sml.png"
"C:\wrox\chapter13\images\06_lrg.png"-"!:\Data\Others\Wrox\images\06_lrg.png"
"C:\wrox\chapter13\images\06_sml.png"-"!:\Data\Others\Wrox\images\06_sml.png"
```

Notice here that the file system for the installation is not specified. The exclamation point (`!`), which is defined before the target path, allows the end user to select which file system to install the content to. For instance, on the Nokia N97, a user could choose to install to the device's memory, to the mass memory, or onto an external memory card (see Figure 13-3).

Flash Lite content is usually installed to the "\Data\Others\" directory by default on the S60 Platform. In Listing 13-6 you can see that an additional subdirectory has been defined, called "Wrox," which will contain both the `imageViewer.swf` and `images.xml` files when the application is installed to the device. A further subdirectory, called "images," is defined for the all the images used in the application. Directories specified as the installation target will be created if they don't already exist.

After you have created a `.pkg` file you can move onto creating the `.sis` file using `makesis.exe`.



FIGURE 13-3

www.it-ebooks.info

Using makesis.exe

The `makesis.exe` tool is an executable file you can run in the Windows command-line tool to generate a `.sis` file. To create the `.sis` file, you simply make a call to the `makesis` command and supply a path to your PKG file source and the output file path for the `.sis` file. The basic syntax looks like this:

```
makesis [PKG_FilePath] [SIS_FilePath]
```

For the `imageViewer.pkg` file example, you can run a command similar to the one shown in Listing 13-7.



LISTING 13-7: Using the MakeSIS command to create imageViewer.sis

```
C:\Program Files\Common Files\Symbian\tools> makesis
C:\wrox\chapter13\imageViewer.pkg C:\wrox\chapter13\imageViewer.sis
```

Depending on the version of your Symbian S60 SDK, you should be able to find the `makesis.exe` file in the tools directory of the installation. Here, the file is referenced in the "C:\Program Files\Common Files\Symbian\tools" folder. Notice that the full paths to the `.pkg` file and target `.sis` file are provided for the command. Both these paths must exist prior to running the command.

Once the command has run, the `.sis` file is generated in the target path, ready for transferring to a compatible device for installing. If, however, you are attempting to install on an S60 3rd edition device you're likely to get the message *"Certificate error. Contact the application supplier."* This relates to the absence of a certificate associated with the application. There's more on this in the next section.

There are other aspects of manually creating the `.sis` file which we haven't discussed, such as providing language options or references to icons.

Later you'll return to the `.pkg` file. In the next section, you examine the process of signing packaged applications; there you'll return to the `.sis` file.

APPLICATION SIGNING

If you consider a device's operating system: How can the device prevent unauthorized access to its capabilities, or to the hardware it resides on? What stops a mobile operator's network from being maliciously attacked? Why should end users be confident that the mobile content they have purchased is actually safe to install?

The purpose of these questions is to get you to think about the main goals of application signing, which essentially aims to give confidence to device manufacturers, mobile operators, online retailers, and the end users that mobile content is safe and that it doesn't contain any malicious code.

www.it-ebooks.info

Certificates

Application signing involves a digital signature, known as a *certificate*, being assigned to an application package. A certificate is commonly purchased from a *certification authority*, or it can be issued by a testing house after the application has been tested against commonly accepted test criteria.

If you want to create commercially available Flash Lite applications, you will need to consider signing for each individual content package you've created, whether it's a `.sis`, `.jar`, or `.cab` file.

Once a certificate has been assigned to an application package, it can be traced to the issuer and so serves as a way to track the originator of the application. The overall purpose of signing is to provide a guarantee that the application can be trusted. There are several types of certificates available, depending on the platform you are targeting your content at.

For testing and development purposes, you can create your own certificate; this is known as *self-signing*. We'll run through this shortly; however, this content is still deemed to be untrusted.

Untrusted Content

By default, any Flash Lite content you package and distribute to the consumer without being signed is deemed *untrusted*. This essential means that during the installation of your application, devices will not be able to verify the integrity of the content or the supplier, because it doesn't have a certificate associated with it.

Typical consequences of *untrusted* content in combination with *unsigned* content appear to be drastic to developers. Safeguards on devices can result in failed installations or prevent an application from accessing some of the device capabilities without notifying the end user first, such as disallowing a simple network connection.

The following message is presented to end users when they try to install content that isn't signed:

> *Unable to install unsigned software. Check 'Software Installation' setting in Application Manager.*

This is the first message that appears when a user attempts installation of an unsigned application on the S60 5th Edition device, the Nokia N97 (see Figure 13-4).



FIGURE 13-4

The installation settings are usually set to "Signed only" on the majority of handsets by default (see Figure 13-5).

Changing the installation setting from "Signed only" to "All" would allow the application installation to progress. Figure 13-6 shows the message that appears once the installation setting has been modified.

www.it-ebooks.info



FIGURE 13-5



FIGURE 13-6

As you can imagine, these messages don't exactly create a good first impression for an application that has just been purchased — neither does it present a welcoming user experience on installation. Nevertheless these "safeguards" are important to maintain integrity of a device and are ultimately there to protect the end user.

Signing applications is a development step you need to implement before going to market. Signing applications can involve expense but is beneficial nonetheless. For testing and distributing freeware, you can simply self-sign applications.

Using makekeys.sis

Earlier you learned how to manually create a `.sis` file using the `makesis.exe` command-line tool. Next, you'll learn how you can create the key and certificate files required to "sign" the application package, using another command-line tool, called `makekeys.exe`. Again, this comes bundled with the Symbian S60 SDK, which once installed can be located in the Symbian tools directory at `C:\Program Files\Common Files\Symbian\tools\makekeys.exe`.

Creating a Certificate and a Private Key

Creating the certificate and key involves specifying several values to five properties:

- **Password** — A defined hidden value associated with the key and certificate.
- **Key length `(len)`** — Represents the size of the file, for example, 1024.

www.it-ebooks.info

- **Distinguished name (dname)** — Consists of five optional subvalues: Contact Name (CN), Organization Unit (OU), Organization (OR), Country (CO), and Email address (EM).
- **Key name** — The name of the key and target path.
- **Certificate name** — The name of the certificate and target path.

Listing 13-8 shows an example of a command to create a certificate and key from the properties previously mentioned, using the `makekeys` tool. Again, you would need to navigate to the tools directory of the Symbian S60 SDK at the command-line tool prompt before running the example:



LISTING 13-8: Using the MakeKeys command to create imageViewer.key and imageViewer.cer

```
C:\Program Files\Common Files\Symbian\tools> makekeys -cert -password password123 -len
1024 -dname "CN=JGAnderson OU=JGAnderson OR=Wrox CO=UK EM=me@JGOrganic.co.uk" C:\wrox\
chapter13\imageViewer.key C:\wrox\chapter13\imageViewer.cer
```

When you run this command, the tool will state that it is "Generating a private key. . . ." In this example the certificate and key files specified in the command are generated. Figure 13-7 shows the `imageViewer.cer` certificate after you double-click on the file to open it. In the window that opens, you'll see the start and end dates for the life of the certificate (see Figure 13-7).



FIGURE 13-7

Using signsis.exe

Once you have created your key and certificate, you can self-sign your application package.

Signing a .sis File

To sign your Flash Lite application, you can use the `signsis.exe` file in the command-line tool, with the following command-line syntax:

```
signsis [unsigned_SIS_FilePath].sis [signed_SIS_FilePath].sisx
[certificate_FilePath].cer [key_FilePath].key [password]
```

Once you run the command, a newly created `.sis` file is created with the certificate, key, and password defined. The extension to the file, `.sisx`, is commonly used to distinguish between an unsigned and, in this case, a self-signed `.sis` file. This `.sis` is self-signed, and can be installed on a supported device. Listing 13-9 shows how the `imageViewer.sisx` file is generated using the `signsis` tool.

www.it-ebooks.info



LISTING 13-9: Using the SignSIS command to self-sign imageViewer.sis

```
C:\Program Files\Common Files\Symbian\tools> signsis
C:\wrox\chapter13\imageViewer.sis C:\wrox\chapter13\imageViewer.sisx
C:\wrox\chapter13\imageViewer.cer C:\wrox\chapter13\imageViewer.key password123
```

Self-Signing Is Signing but Still Untrusted

It's common across many device manufacturers and mobile operators only to allow signed applications to be installed. For distribution via aggregators and operator networks, it is necessary to purchase certificates and sign your applications. Failure to do that will result in failed installations, and you could potentially end up having to make lots of refunds.

Signing for Windows Mobile

The steps for signing `.cab` files are very similar to those for signing `.sis` files. Instead of using the Symbian SDK, you need to use the Microsoft .NET Framework SDK, which can be downloaded at `http://msdn.microsoft.com/en-us/netframework/aa731542.aspx`. You then use the `makecert`, `cert2spc`, and `signtool` command-line tools, in a similar fashion to signing .sis files, to create keys and certificates, and then validate the package.

For more information visit the "Steps for signing a .cab file" Web page at `http://support.microsoft.com/kb/247257`.

In order to sign `.cab` files for commercial distribution, you need to sign up with a certification authority, such as Thawte (`http://www.thawte.com/`) or VeriSign (`http://www.verisign.com/`), that can provide the digital certificates required for signing applications.

Both companies require you to purchase a certificate for signing applications, and in return you receive your application certificate IDs, which you merge with your packaged `.cab` files.

Once you've received your IDs, you then use the `signtool.exe` to assign the certificate to a package.

Signing for Symbian S60

Symbian Signed is the signing program that's administered by the Symbian Foundation on behalf of the community.

To deploy your Flash Lite applications commercially, you need to purchase a publisher ID and an individual content ID for each application. Only applications validly signed with a Publisher ID Certificate are allowed to be submitted for signing, in order to become an Express Signed Application or Certified Signed Application.

Signing for Java

Earlier you learned how to package `.jar` files using Swf2Jar. Signing `.jar` files is also similar to signing the `.sis` files. You need to use the Java Development Kit (JDK), which can be downloaded from `http://java.sun.com/javase/downloads/index.jsp`. Here, you'll find the `jarsigner` and

www.it-ebooks.info

keytool command-line tools. For more information, check out the Sun's JAR signing Web site page at http://java.sun.com/docs/books/tutorial/deployment/jar/signing.html.

Signing Programs

During development you can register with a signing program to help you get your applications signed with trusted certificates for commercial release.

Symbian Signed

The Symbian community uses the Symbian Signed program for applications packaged in .sis files. At the Web site there are test criteria that you can use to validate your applications. The Symbian Signed Test Criteria can be found on the Symbian Developer Web site (http://developer.symbian.org/wiki/images/f/fc/Symbian_Signed_Test_Criteria_3.0.3.pdf).



Symbian and other platforms have a platform security model implemented in their operating systems that effectively determines whether an application can have access to specific APIs considered to be sensitive and also whether restricted areas of a file system can be accessed. The model uses digital certificates assigned to an application during the signing process to establish this.

We're not going to go through test criteria here or the process of signing your applications. For more information on signing applications for the Symbian platform, you should visit the Symbian developer's Web site (http://developer.symbian.org/) and the Symbian Signed Web site (https://www.symbiansigned.com/app/page).

Publishing to Aggregators

There are a number of content aggregators with respectable names waiting to sell your content:

- **GetJar** — http://www.getjar.com/
- **Thumbplay** — http://www.thumbplay.com/
- **Zed** — http://www.zed.com/
- **Handandgo** — http://www.handango.com/

APPLICATION ICONS

Application icons are a crucial part of the packaging process, as they should be what the user sees after they install your application to their device. Figure 13-8 shows an screen shot taken of the grid view highlighting the Calendar application for the Nokia N97.

Figure 13-9 shows a screen shot taken of the list view, highlighting the Calendar application for the Nokia N97.

www.it-ebooks.info



FIGURE 13-8



FIGURE 13-9

You've already covered the Project Capuchin method for including icons using the Swf2Jar tool. For Symbian S60 3rd Edition, icons must be tied to the registration of the application in the `.pkg` file.

There are a number of tools for this process that you will need to use to achieve this. First, there is svg2svgt, a tool that converts the Scalable Vector Graphics (SVG, `http://www.w3.org/Graphics/SVG/`) format into the SVG Tiny format, a subset of SVG, which is used so that unsupported nodes within the source are not in the file and to ensure the file is rendered on the device the way it was on the PC. The installation for the tool can be found in the Symbian S60 SDK `\S60Tools\svg2svgt\installer` path.

There are also two tools that you need to run via the command line, called `Mifconv` and `Epocrc`, both of which can be found in the \epoc32\tools folder of the Symbian S60 SDK.

Using mifconv.exe

The Mifconv tool generates icon files in Multi-Icon File (MIF, `.mif`) format from SVG and bitmap (BMP, `.bmp`) source files. After converting your `.svg` file, you can create the `.mif` file using the command line, as shown in Listing 13-10, which will generate the `imageViewer.mif` file from an SVG file called `icon.svg`.



LISTING 13-10: Using the MifConv tool to generate a imageViewer.mif

```
C:\Symbian\9.1\S60_3rd\Epoc32\tools> mifconv imageViewer.mif icon.svg
```

www.it-ebooks.info

The .mif file now needs to be referenced by a resource file associated with the application. For this, you use the resource builder tool called epocrc.exe.

Using epocrc.exe

The epocrc.exe tool converts a registration source (.rss) file to a compiled .rsc file, which contains references to resources. The compiled resource files are made available to an application at runtime, and must be referenced in the .pkg file for the application installer.

Before using the tool, let's take a look at two resource files you would need to define: the localizable registration information and the application's registration information.

Defining the Localizable Registration Information

The following code snippet shows the source for the localizable registration information in a file called imageViewer.rss. Here, information such as the caption label and icon are specified for the application. Notice that the value assigned to the icon_file property will be the path to the .mif file on the device when it is installed.




```
#include <appinfo.rh>

RESOURCE LOCALISABLE_APP_INFO 42780
{
        short_caption = "imageViewer";
        caption_and_icon =
        {
                CAPTION_AND_ICON_INFO
                {
                        caption = "Chapter 8 Image Viewer";
                        number_of_icons = 1;
                        icon_file = "\\resource\\apps\\imageViewer.mif";
                }
        };
        view_list = VIEW_DATA{};
}
```

code snippet imageViewer.rss

Defining the Application's Registration Information

The following code snippet shows the registration resource information in the file called imageViewer_reg.rsc. This defines information such as the application's file name, the name and path of the localizable registration information file, and the UID. The value assigned to the property localizable_resource_file represents the path on the device where the localizable registration information will reside. Notice here that the target installation drive and the file extension (.rsc) are not specified.

www.it-ebooks.info




```
#include <appinfo.rh>

UID2 KUidAppRegistrationResourceFile
UID3 0xE0000001

RESOURCE APP_REGISTRATION_INFO
{
      app_file="imageViewer";
      localisable_resource_file = "\\resource\\apps\\imageViewer";
      localisable_resource_id = 42780;
      datatype_list = DATATYPE{};
      file_ownership_list = FILE_OWNERSHIP_INFO{};
      service_list = SERVICE_INFO{};
}
```

code snippet imageViewer_reg.rss

Once these files have been defined, you can use the `epocrc.exe` tool to generate the compiled registration files. Listing 13-11 shows the compilation of `imageViewer_reg.rss` to `imageViewer_reg.rsc`. In addition to the input file `imageViewer_reg.rss`, you can see there are three optional parameters that are used on the command line:

- `-uid3`, which specifies the UID for the application.
- `-o`, which specifies the output file.
- `-I`, which specifies the include path for the command.

In both registration source files, you'll notice the inclusion of the `appinfo.rh` file. The included path is where this file resides, so this should also be specified in the command. Here, the source and output are in the same directory as the `epocrc.exe` tool.



LISTING 13-11: Using the Epocrc tool to create the application's registration information file for imageViewer.

```
C:\Symbian\9.1\S60_3rd\Epoc32\tools> epocrc -uid3 0xE0000001 imageViewer_reg.rss
-o"imageViewer_reg.rsc" -I C:\Symbian\9.1\S60_3rd\Epoc32\include
```

Having compiled the resource files, they can now be included in the `.pkg` file so that the `.sis` file can be created and then re-signed.

Returning to the `.pkg` file, you would add the `imageViewer_reg.rsc`, `imageViewer.rsc`, and `imageViewer.mif` files to the installation, as shown in Listing 13-12. Notice that the `imageViewer.rsc` and `imageViewer.mif` files are installed to the "\resource\apps" directory, as specified in the registration information. The application's registration file needs to be installed in the "\private\10003a3f\import\apps\" directory on the device.

www.it-ebooks.info



LISTING 13-12: Including the registration and resource files in imageViewer.pkg

```
&EN
#{"imageViewer"},(0xE0000001),1,0,0
[0x102752AE],0,0,0,{"S60ProductID"}
%{"JGAnderson"}
:"JGAnderson"
"C:\wrox\chapter13\imageViewer.swf"-"!:\Data\Others\Wrox\imageViewer.swf"
"C:\wrox\chapter13\images.xml"-"!:\Data\Others\Wrox\images.xml"
"C:\wrox\chapter13\images\01_lrg.png"-"!:\Data\Others\Wrox\images\01_lrg.png"
"C:\wrox\chapter13\images\01_sml.png"-"!:\Data\Others\Wrox\images\01_sml.png"
"C:\wrox\chapter13\images\02_lrg.png"-"!:\Data\Others\Wrox\images\02_lrg.png"
"C:\wrox\chapter13\images\02_sml.png"-"!:\Data\Others\Wrox\images\02_sml.png"
"C:\wrox\chapter13\images\03_lrg.png"-"!:\Data\Others\Wrox\images\03_lrg.png"
"C:\wrox\chapter13\images\03_sml.png"-"!:\Data\Others\Wrox\images\03_sml.png"
"C:\wrox\chapter13\images\04_lrg.png"-"!:\Data\Others\Wrox\images\04_lrg.png"
"C:\wrox\chapter13\images\04_sml.png"-"!:\Data\Others\Wrox\images\04_sml.png"
"C:\wrox\chapter13\images\05_lrg.png"-"!:\Data\Others\Wrox\images\05_lrg.png"
"C:\wrox\chapter13\images\05_sml.png"-"!:\Data\Others\Wrox\images\05_sml.png"
"C:\wrox\chapter13\images\06_lrg.png"-"!:\Data\Others\Wrox\images\06_lrg.png"
"C:\wrox\chapter13\images\06_sml.png"-"!:\Data\Others\Wrox\images\06_sml.png"
"C:\wrox\chapter13\imageViewer.rsc"-"!:\resources\apps\imageViewer.rsc"
"C:\wrox\chapter13\imageViewer.mif"-"!:\resources\apps\imageViewer.mif"
"C:\wrox\chapter13\imageViewer_reg.rsc"-"!:\private\10003a3f\import\apps\
imageViewer_reg.rsc"
```

After re-running the Makesis and Signsis commands from earlier, you will have covered how to include an application icon with your Flash Lite application for the Symbian S60 3rd Edition.

Given that this is rather a complex way to include an application icon for your application, there are a number of resources at the Symbian Developer and Nokia Forum that will help should you get stuck in a rut.

NOKIA FLASH PACKAGING TOOL

The Nokia Flash Packaging Tool is an online packaging service that provides another route to packaging content targeted at Symbian S60 devices. The tool is a popular addition to Forum Nokia for S60-based developers. In addition, the service has an option for you to package content for the Symbian Series 40 platform. If you want to create content for Series 40 devices, you need to create a Nokia Flash Lite (NFL) package.

I recommend visiting the Forum Nokia site if you want to learn more on Series 40: `http://www.forum.nokia.com/Technology_Topics/Device_Platforms/Series_40/`.

While the tool is aimed at Nokia devices, downloading and installing the Symbian S60 SDK is not required, and no additional setup is required. The tool works simply as a Web application.

www.it-ebooks.info

The tool allows you to create packaged content in three distinct forms:

- **Application** — Supply the Application Name, Source files, Icon, Installation notes, and your main Source file.
- **Advanced** — Supply the Short caption, Target Directory, Version number, Vendor Name, and Language.
- **Signing** — Supply the Certificate, Private Key, Passphrase, and UID.

Using the tool provides a much faster alternative to manually creating `.sis` files. Next, you'll be introduced to some of the distribution channels available to Flash developers for monetizing content.

INTRODUCING OVI FROM NOKIA

Ovi (`http://www.ovi.com`), *the Finnish word for "door,"* is Nokia's Internet services gateway, which gives registered users access to a range of functions and services on the go and over the air.

The current list of Ovi services includes the following:

- **Ovi Files** — Access folders and files residing on your home computer, from another device or computer with a Web browser through Ovi.com. In addition you can store the files online securely.
- **Ovi Mail** — Send email from your handset using a registered Ovi.com email address.
- **Ovi Maps** — Browse maps and save them to your Ovi account.
- **Ovi Share** — Upload photos and video content from your device to share on your Ovi.com space.
- **Ovi Sync** — Sync contacts, calendar entries, and notes between devices, your computer, and Ovi.com.
- **Ovi Store** — Allows users to browse and download video, music, ringtones, applications, and games.

The majority of the features listed allow you to seamlessly stay connected between your computer and your mobile device and allow you to discover and share content, through a gateway that has millions of users worldwide. With the Ovi Store, you can distribute your Flash-enabled mobile content worldwide and generate revenue.

The Ovi Store

The Ovi Store is a one-stop global marketplace for consumers of mobile content developed for Nokia devices. It offers both free and paid-for content, giving Flash developers around the world the opportunity to publish, sell, and distribute content to the consumer.

As a consumer, you can access mobile content through your Ovi account. One of the key features of the store is being able to send MMS links to an end user, facilitating the downloading of content straight to the device.

www.it-ebooks.info

Become an Ovi Publisher

To distribute your Flash-enabled mobile content to the millions of Nokia consumers in Ovi, you need to become a publisher and sign-up for a Publisher's Account. Applications also need to be signed. Register as a publisher at `http://publish.ovi.com`.

INTRODUCING PLAYNOW ARENA FROM SONY ERICSSON

PlayNow arena (`http://www.playnow-arena.com`) is a Sony Ericsson portal for accessing multimedia, offering free and paid-for content downloads, including music, games, ringtones, themes, wallpapers, and applications.

This is another place where you can sell your completed Flash-enabled mobile content to end users who are looking to access a full range of free and premium content via their mobile phone or computer from Sony Ericsson.

Sony Ericsson Content Submission

To submit content you need to register on the Sony Ericsson Developer World site at `http://developer.sonyericsson.com`. From there, you can navigate to the Sony Ericsson Content Submission site (`https://submit.sonyericsson.com/`), where you will be provided with links to register for submitting applications. The site also provides you with specific details for submitting content, including quality guidelines and the actual submission procedure, which is discussed next.

Submissions Procedure

The Sony Ericsson submissions procedure is a four-step process, which includes:

1. **Submission** — Submit your content to Sony Ericsson.
2. **Review** — Sony Ericsson reviews your content.
3. **Certify** — You ensure that the application is signed.
4. **Deployment** — Publishing your certified application.

For the submission to PlayNow arena, you have the choice of submitting the Flash Lite application's packaged content as a JAR or SIS file.

Distribution is currently open to about 70 markets around the world, where you can earn 70% of the net revenue from sales of your applications.

The submission guidelines also provide details on a range of *Supported Business Models for Deployment*, including:

- Free (content without advertising)
- Free content (with advertising)
- Pay once/pay-to-download

www.it-ebooks.info

- Try & buy content
- Free basic features; premier features at a cost
- Periodic recurring charges content



As of July 2009, PlayNow arena is available in 70 countries and is compatible with 38 Sony Ericsson phone models, which equates to 25 million potential customers for your application. Not all of these devices support Project Capuchin or Flash Lite, but this gives you an idea of the potential for distribution as Sony Ericsson increases the number of supported Flash Lite and Project Capuchin devices.

SUMMARY

On the whole, packaging Flash Lite content seems complex from the outset, albeit a process made a whole lot easier with automated tools. However, when the process is understood and you've learned about the commands, tools, and files you need to create for the manual process, it's a piece of cake . . . isn't it?

In this chapter, you learned how you can package Flash Lite applications, ready for distributing to the consumer for both the Symbian S60 and Java platforms, covering topics such as swf2Jar, `.sis` file creation, and application signing. You were also introduced to places where you can sign packages, and the key aggregators and marketplaces where you distribute your Flash Lite content to the consumer.

www.it-ebooks.info

APPENDIX

Flash Lite 3.x ActionScript 2.0 Quick Reference: From Array to XMLSocket

In this appendix, you'll find a quick reference to all the class methods, properties, operators, and events found in the ActionScript 2.0 Library that are supported in Flash Lite up to version 3.1.

The reference is taken from the *Adobe Flash Lite 2.x and 3.x ActionScript Language Reference* found at `http://help.adobe.com/en_US/FlashLite/2.0_FlashLiteAPIReference2/index.html`.

ARRAY

CONSTRUCTOR

```
Array([value:Object])
```

PROPERTIES

```
Length:Number
```

www.it-ebooks.info

METHODS

```
concat([value:Object]):Array
join([delimiter:String):String
pop():Object
push(value:Object):Number
reverse():Void
shift():Object
slice([startIndex:Number], [endIndex:Number]):Array
sort([compareFunction:Object], [options:Object]):Array
sortOn(fieldname:Object, [options:Object]):Array
splice(startIndex:Number, deleteCount:Number], [value:Object]):Array
toString():String
unshift(value:Object):Number
```

CONSTANTS

```
CASEINSENSITIVE:Number
DESCENDING:Number
NUMERIC:Number
RETURNINDEXEDARRAY:
Number
UNIQUESORT:Number
```

BOOLEAN

CONSTRUCTOR

```
Boolean([value:Object])
```

METHODS

```
toString():String
valueOf():Boolean
```

www.it-ebooks.info

BUTTON

PROPERTIES

```
_alpha:Number
enabled:Boolean
_focusrect:Boolean
_height:Boolean
_highquality:Number
_name:String
_parent:MovieClip
_quality:String
_rotation:Number
_soundbuftime:Number
tabEnabled:Boolean
tabIndex:Number
_target:String
trackAsMenu:Boolean
_url:String
_visible:Boolean
_width:Number
_x:Number
_xmouse:Number
_xscale:Number
_y:Number
_ymouse:Number
_yscale:Number
```

METHODS

```
getDepth():Number
```

EVENTS

```
onDragOut = function () {}
onDragOver = function () {}
onKeyDown = function () {}
onKeyUp = function () {}
onKillFocus = function (newFocus:Object) {}
onPress = function () {}
onRelease = function () {}
onReleaseOutSide = function () {}
onRollOut = function () {}
onSetFocus = function (oldFocus:Object) {}
```

www.it-ebooks.info

COLOR

CONSTRUCTOR

```
Color(target:Object)
```

METHODS

```
getRGB():Number
getTransform():Object
setRGB(offset:Number):Void
setTransform(transformObject:Object):Void
```

DATE

CONSTRUCTOR

```
Date([yearOrTimevalue:Number], [month:Number], [date:Number], [hour:Number],
[minute:Number], [second:Number], [millisecond:Number])
```

METHODS

```
getDate():Number
getDay():Number
getFullYear():Number
getHours():Number
getLocaleLongDate():String
getLocaleShortDate():String
getLocaleTime():String
getMilliseconds():Number
getMinutes():Number
getMonth():Number
getSeconds():Number
getTime():Number
getTimezoneOffset():Number
getUTCDate():Number
getUTCDay():Number
getUTCFullYear():Number
getUTCHours():Number
getUTCMilliseconds():Number
getUTCMinutes():Number
getUTCMonth():Number
getUTCSeconds():Number
```

www.it-ebooks.info

METHODS

```
getUTCYear():Number
getYear():Number
setDate(date:Number):Number
setFullYear(year:Number, [month:Number], [date:Number]):Number
setHours(hour:Number):Number
setMilliseconds(millisecond:Number):Number
setMinutes(minute:Number):Number
setMonth(month:Number, [date]):Number
setSeconds(seconds:Number):Number
setTime(millisecond:Number):Number
setUTCDate(date:Number):Number
setUTCFullYear(year:Number, [month:Number], [date:Number]):Number
setUTCHours(hour:Number, [minute:Number],
            [second:Number], [millisecond:Number]):Number
setUTCMilliseconds(millisecond:Number):Number
setUTCMinutes(minute:Number, [second:Number], [millisecond:Number]):Number
setUTCMonth(month:Number, [date:Number]):Number
setUTCSeconds(seconds:Number, [millisecond:Number]):Number
setYear(year:Number):Number
toString():String
UTC(year:Number, month:Number, [date:Number], [hour:Number], [minute:Number],
[second:Number], [millisecond:Number]):Number
valueOf():Number
```

ERROR

CONSTRUCTOR

```
Error([message:String])
```

METHODS

```
toString():String
```

PROPERTIES

```
message:String
name:String
```

www.it-ebooks.info

EXTENDEDKEY

PROPERTIES

```
SOFT1:String
SOFT2:String
SOFT3:String
SOFT4:String
SOFT5:String
SOFT6:String
SOFT7:String
SOFT8:String
SOFT9:String
SOFT10:String
SOFT11:String
SOFT12:String
```

FLASH.DISPLAY.BITMAPDATA

CONSTRUCTOR

```
BitmapData(width:Number, height:Number, [transparent:Boolean], [fillColor:
Number])
```

METHODS

```
clone():BitmapData
colorTransform(rect:Rectangle, colorTransform:ColorTransform):Void
copyChannel(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point,
sourceChannel:Number, destChannel:Number):Void
copyPixels(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point,
[alphaBitmap:BitmapData], [alphaPoint:Point, [mergeAlpha:Boolean]):Void
dispose():Void
draw(source:Object, [matrix:Matrix], [colorTransform:ColorTransform],
 [blendMode:Object], [cliprect:Rectangle], [smooth:Boolean]):Void
fillRect(rect:Rectangle, color:Number):Void
floodFill(x:Number, y:Number, color:Number):Void
getColorBoundsRect(mask:Number, color:Number, [findColor:Boolean]):Rectangle
getPixel(x:Number, y:Number):Number
getPixel32(x:Number, y:Number):Number
hitTest(firstPoint:Point, firstAlphaThreshold:Number, secondObject:Object,
[secondBitmapPoint:Point], [secondAlphaThreshold:Number]):Boolean
```

www.it-ebooks.info

METHODS

```
loadBitmap(id:String):BitmapData
merge(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point,
redMult:Number, greenMult:Number, blueMult:Number, alphaMult:Number):Void
setPixel(x:Number, y:Number, color:Number):Void
setPixel32(x:Number, y:Number, color:Number):Void
```

PROPERTIES

```
height:Number
rectangle:Rectangle
transparent:Boolean
width:Number
```

FLASH.GEOM.COLORTRANSFORM

CONSTRUCTOR

```
ColorTransform([redMultiplier:Number],[greenMultiplier:Number],
[blueMultiplier:Number], [alphaMultiplier:Number], [redOffset:
Number],[greenOffset:Number],[blueOffset:Number],[alphaOffset:Number])
```

PROPERTIES

```
alphaMultiplier:Number
alphaOffset:Number
blueMultiplier:Number
blueOffset:Number
greenMultiplier:Number
greenOffset:Number
redMultiplier:Number
redOffset:Number
rgb:Number
```

METHODS

```
concat(second:ColorTransform):Void
toString():String
```

www.it-ebooks.info

FLASH.GEOM.MATRIX

PROPERTIES

```
a:Number
b:Number
c:Number
d:Number
tx:Number
ty:Number
```

METHODS

```
clone():Matrix
concat(m:Matrix):Void
createBox(scaleX:Number, scaleY:Number,
          [rotation:Number], [tx:Number], [ty:Number]):Void
createGradientBox(width:Number, height:Number,
          [rotation:Number],  [tx:Number], [ty:Number]):Void
detlaTransformationPoint(pt:Point):Point
identity():Void
invert():Void
rotate(angle:Number):Void
scale(sx:Number, sy:Number):Void
toString():String
transformPoint(pt:Point):Point
translate(tx:Number, ty:Number):Void
```

FLASH.GEOM.POINT

CONSTRUCTOR

```
Point(x:Number,y:Number)
```

PROPERTIES

```
length:Number
x:Number
y:Number
```

www.it-ebooks.info

METHODS

```
add(v:Point):Point
clone():Point
distance(pt1:Point, pt2:Point, f:Number):Point
equals(toCompare:Object):Boolean
interpolate(pt2:Point, pt2:Point, f:Number):Point
normalize(length:Number):Void
offset(dx:Number, dy:Number):Void
polar(len:Number, angle:Number):Point
substract(v:Point):Point
toString():String
```

FLASH.GEOM.RECTANGLE

CONSTRUCTOR

```
Rectangle(x:Number,y:Number,width:Number,height:Number)
```

PROPERTIES

```
bottom:Number
bottomright:Point
height:Number
left:Number
right:Number
size:Point
top:Number
topLeft:Point
width:Number
x:Number
y:Number
```

METHODS

```
clone():Rectangle
contains(x:Number, y:Number):Boolean
containsPoint(pt:Point):Boolean
containsRectangle(rect:Rectangle):Boolean
equals(toCompare:Object):Boolean
inflate(dx:Number, dy:Number):Void
inflatePoint(pt:Point):Void
intersection(toIntersect:Rectangle):Rectangle
```

www.it-ebooks.info

METHODS

```
intersects(toIntersect:Rectangle):Boolean
isEmpty():Boolean
offset(dx:Number, dy:Number):Void
offsetPoint(pt:Point):Void
setEmpty():Void
toString():String
union(toUnion:Rectangle):Rectangle
```

FLASH.GEOM.TRANSFORM

CONSTRUCTOR

```
Transform(mc:MovieClip)
```

PROPERTIES

```
colorTransform:ColorTransform
concatenatedColorTransform:ColorTransform
concatenatedMatrix:Matrix
matrix:Matrix
pixelBounds:Rectangle
```

FUNCTION

METHODS

```
apply(thisObject:Object, [argArray:Array])
call(thisObject:Object, [parameter1:Object])
```

GLOBAL

FUNCTIONS

```
Array([numElements], [elementN]):Array
Boolean(expression:Object):Boolean
clearInterval(intervalID:Number)
duplicateMovieClip(target:Object, newname:String, depth:Number)
```

www.it-ebooks.info

FUNCTIONS

```
escape(expression:String):String
eval(expression:Object):Object
fscommand(command:String, parameters:String)
fscommand2(command:String, parameters:String)
getTimer():Number
getURL(url:String, [window:String], [method:String])
getVersion():String
gotoAndPlay([scene:String], frame:Object)
gotoAndStop([scene:String], frame:Object)
isFinite(expression:Object):Boolean
isNaN(expression:Object):Boolean
loadMovie(url:String, target:Object, [method:String])
loadMovieNum(url:String, level:Number, [method:String])
loadVariables(url:String, target:Object, [method:String])
loadVariablesNum(url:String, level:Number, [method:String])
nextFrame()
nextScene()
Number(expression:Object):Number
Object([value:Object]):Object
on(mouseEvent:Object)
onClipEvent(movieEvent:Object)
parseFloat(string:String):Number
parseInt(expression:String, [radix:Number]):Number
play()
prevFrame()
prevScene()
removeMovieClip(target:Object)
setInterval(functionName:Object, interval:Number, [param:
Object], objectName:Object, methodName:String):Number
setProperty(target:Object, property:Object, expression:Object)
startDrag(target:Object, [lock:Boolean],
[left,top,right,bottom:Number])
stop()
stopAllSounds()
stopDrag()
String(expression:Object):String
targetPath(targetObject:Object):String
trace(expression:Object)
unescape(string:String):String
unloadMovie(target)
unloadMovieNum(level:Number)
```

www.it-ebooks.info

KEY

PROPERTIES

```
BACKSPACE:Number
CAPSLOCK:Number
CONTROL:Number
DELETEKEY:Number
DOWN:Number
END:Number
ENTER:Number
ESCAPE:Number
HOME:Number
INSERT:Number
LEFT:Number
_listeners:Array
PGDN:Number
PGUP:Number
RIGHT:Number
SHIFT:Number
SPACE:Number
TAB:Number
UP:Number
```

METHODS

```
addListener(listener:Object):Void
getAscii():Number
getCode():Number
isDown(code:Number):Boolean
removeListener(listener:Object):Boolean
```

EVENTS

```
onKeyDown = function () {}
onKeyUp = function () {}
```

LOADVARS

CONSTRUCTOR

```
LoadVars()
```

www.it-ebooks.info

PROPERTIES

```
contentType:String
loaded:Boolean
```

METHODS

```
addRequestHeader(header:Object, headerValue:String):Void
decode(queryString:String):Void
getBytesLoaded():Number
getBytesTotal():Number
load(url:String):Number
send(url:String, target:String, [method:String]):Boolean
sendAndLoad(url:String, target:Object, [method:String]):Boolean
toString():String
```

EVENTS

```
onData = function (src:String) {}
onLoad = function (success:Boolean) {}
```

LOCALCONNECTION

CONSTRUCTOR

```
LocalConnection()
```

METHODS

```
close():Void
connect(connectionName:String):Boolean
domain():String
send(connectionName:String, methodName:String, [args:Object]):Boolean
```

EVENTS

```
allowDomain = function([sendingDomain:String]) {}
allowInsecureDomain = function([sendingDomain:String]) {}
onStatus = function(infoObject:Object) {}
```

www.it-ebooks.info

MATH

PROPERTIES

```
E:Number
LN10:Number
LN2:Number
LOG10E:Number
LOG2E:Number
PI:Number
SQRT1_2:Number
SQRT2:Number
```

METHODS

```
abs(x:Number):Number
acos(x:Number):Number
asin(x:Number):Number
atan(tangent:Number):Number
atan2(y:Number, x:Number):Number
ceil(x:Number):Number
cos(x:Number):Number
exp(x:Number):Number
floor(x:Number):Number
log(x:Number):Number
max(y:Number, x:Number):Number
min(y:Number, x:Number):Number
pow(y:Number, x:Number):Number
random(x:Number):Number
round(x:Number):Number
sin(x:Number):Number
sqrt(x:Number):Number
tan(x:Number):Number
```

MOUSE

METHODS

```
addListener(listener:Object):Void
removeListener(listener:Object):Boolean
```

www.it-ebooks.info

EVENTS

```
onMouseDown = function() {}
onMouseMove = function() {}
onMouseUp = function() {}
```

MOVIECLIP

PROPERTIES

```
_alpha:Number
_currentframe:Number
_droptarget:String
enabled:Boolean
focusEnabled:Boolean
_focusrect:Boolean
_framesloaded:Number
_height:Number
hitArea:Object
_highquality:Number
_lockroot:Boolean
_name:String
_parent:MovieClip
_quality:String
_rotation:Number
_soundbuftime:Number
tabChildren:Boolean
tabEnabled:Boolean
tabIndex:Number
_target:String
_totalframes:Number
trackAsMenu:Boolean
_url:String
_visible:Boolean
_width:Number
_x:Number
_xmouse:Number
_xscale:Number
_y:Number
_ymouse:Number
_yscale:Number
```

www.it-ebooks.info

METHODS

```
attachMovie(id:string, name:String, depth:Number, [initObject:Object]):
MovieClip
beginFill(rgb:Number, [alpha:Number]):Void
beginGradientFill(fillType:String, colors:Array, alphas:Array, ratios:Array,
matrix:Object):Void
clear():Void
createEmptyMovieClip(name:String, depth:Number):MovieClip
createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:
Number, height:Number):TextField
curveTo(controlX:Number, controlY:Number, anchor:Number, anchorY:Number):Void
duplicateMovieClip(name:String, depth:Number, [initObject:Object]):MovieClip
endFill():Void
getBounds(bounds:Object):Object
getBytesLoaded():Number
getBytesTotal():Number
getDepth():Number
getInstanceAtDepth(depth:Number):MovieClip
getNextHighestDepth():Number
getSWFVersion():Number
getURL(url:String, [window:String], [method:String]):Void
globalToLocal(pt:Object):Void
gotoAndPlay(frame:Object):Void
gotoAndStop(frame:Object):Void
hitTest():Boolean
lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean,
noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number):Void
lineTo(x:Number, y:Number):Void
loadMovie(url:String, [method]):Void
loadVariables(url:String, [method:String]):Void
localToGlobal(pt:Object):Void
moveTo(x:Number, y:Number):Void
nextFrame():Void
play():Void
prevFrame():Void
removeMovieClip():Void
setMask(mc:Object):Void
startDrag([lockCenter:Boolean], [left:Number], [top:Number], [right:Number],
[bottom:Number]):Void
stop():Void
stopDrag():Void
swapDepths(target:Object):Void
unloadMovie():Void
```

www.it-ebooks.info

EVENT

```
onData = function() {}
onDragOut = function() {}
onDragOver = function() {}
onEnterFrame = function() {}
onKeyDown = function() {}
onKeyUp = function() {}
onKillFocus = function(newFocus:Object) {}
onLoad = function() {}
onMouseDown = function() {}
onMouseMove = function() {}
onMouseUp = function() {}
onPress = function() {}
onRelease = function() {}
onReleaseOutside = function() {}
onRollOut = function() {}
onRollOver = function() {}
onSetFocus = function(oldFocus:Object) {}
onUnload = function() {}
```

MOVIECLIPLOADER

CONSTRUCTOR

```
MovieClipLoader()
```

METHODS

```
addListener(listener:Object):Boolean
getProgress(target:Object):Object
loadClip(url:String, target:Object):Boolean
removeListener(listener:Object):Boolean
unloadClip(target:Object):Boolean
```

EVENTS

```
onLoadComplete = function(listenerObject, [target_mc]) {}
onLoadError = function(listenerObject, errorCode) {}
onLoadInit = function([target_mc]) {}
onLoadProgress = function([target_mc], loadedBytes, totalBytes) {}
onLoadStart = function([target_mc]) {}
```

www.it-ebooks.info

NETCONNECTION

CONSTRUCTOR

```
NetConnection()
```

METHODS

```
connect():Void
close():Void
```

NETSTREAM

CONSTRUCTOR

```
NetStream()
```

PROPERTIES

```
bufferLength:Number
bufferTime:Number
bytesLoaded:Number
bytesTotal:Number
currentFps:Number
time:Number
```

METHODS

```
close():Void
pause([flag:Boolean]):Void
play(name:String, [start:Number], [len:Number], [reset:Object]):Void
seek(offset:Number):Void
setBufferTime(bufferTime:Number):Void
```

EVENTS

```
onStatus = function(infoObject:Object) {}
onCuePoint = function(infoObject:Object) {}
onMetaData = function(infoObject:Object) {}
```

www.it-ebooks.info

NUMBER

CONSTRUCTOR

```
Number(num:Object)
```

PROPERTIES

```
MAX_VALUE:Number
MIN_VALUE:Number
NaN:Number
NEGATIVE_INFINITY:Number
POSITIVE_INFINITY:Number
```

METHODS

```
toString(radix:Number):String
valueOf():Number
```

OBJECT

CONSTRUCTOR

```
Object()
```

PROPERTIES

```
constructor:Object
__proto__:Object
Prototype:Object
__resolve:Object
```

METHODS

```
addProperty(name:String, getter:Function, setter:Function):Boolean
hasOwnProperty(name:String):Boolean
isPropertyEnumerable(name:String):Boolean
isPrototypeOf(theClass:Object):Boolean
registerClass(name:String, theClass:Function):Boolean
toString():String
unwatch(name:String):Boolean
valueOf():Object
watch(name:String, callback:Function, [userData:Object]):Boolean
```

www.it-ebooks.info

SELECTION

METHODS

```
addListener(listener:Object):Void
getFocus():String
removeListener(listener:Object):Boolean
setFocus(newFocus:Object):Boolean
setSelection(beginIndex:Number, endIndex:Number):Void
```

EVENT

```
onSetFocus = function([oldFocus], [newFocus]) {}
```

SHAREDOBJECT

PROPERTIES

```
data:Object
```

METHODS

```
addListener(objectName:String, notifyFunction:Function):Void
clear():Void
flush(minDiskSpace:Number):Object
getLocal(name:String):SharedObject
getMaxSize():Number
getSize():Number
removeListener(objectName:String):Void
```

EVENTS

```
onStatus = function(infoObject:Object) {}
```

www.it-ebooks.info

SOUND

CONSTRUCTOR

```
Sound([target:Object])
```

PROPERTIES

```
duration:Number
id3:Object
position:Number
```

METHODS

```
attachSound(id:String):Void
getBytesLoaded():Number
getBytesTotal():Number
getPan():Number
getTransform():Object
getVolume():Number
loadSound(url:String, isStreaming:Boolean):Void
setPan(value:Number):Void
setTransform(transformObject:Object):Void
setVolume(value:Number):Void
start([secondOffset:Number], [loops:Number]):Void
stop([linkageID:String]):Void
```

EVENTS

```
onId3 = function() {}
onLoad = function(success:Boolean) {}
onSoundComplete = function() {}
```

STAGE

PROPERTIES

```
align:String
height:Number
scaleMode:String
width:Number
```

www.it-ebooks.info

METHODS

```
addListener(listener:Object):Void
removeListener(listener:Object):Void
```

EVENT

```
onResize = function() {}
```

STRING

CONSTRUCTOR

```
String(value:String)
```

PROPERTIES

```
length:Number
```

METHODS

```
charAt(index:Number):String
charCodeAt(index:Number):Number
concat(value:Object):String
fromCharCode():String
indexOf(value:String, [startIndex:Number]):Number
lastIndexOf(value:String, [startIndex:Number]):Number
slice(start:Number, end:Number):String
split(delimiter:String), [limit:Number]):Array
substr(start:Number, length:Number):String
substring(start:Number, end:Number):String
toLowerCase():String
toString():String
toUpperCase():String
valueOf():String
```

www.it-ebooks.info

SYSTEM.CAPABILITIES

PROPERTIES

```
audioMIMETypes:Array
avHardwareDisable:Boolean
has4WayKeyAS:Boolean
hasAccessibility:Boolean
hasAudio:Boolean
hasAudioEncoder:Boolean
hasCMIDI:Boolean
hasCompoundSound:Boolean
hasDataLoading:Boolean
hasEmail:Boolean
hasEmbeddedVideo:Boolean
hasMappableSoftKeys:Boolean
hasMFI:Boolean
hasMIDI:Boolean
hasMP3:Boolean
hasPrinting:Boolean
hasQwertyKeyboard:Boolean
hasScreenBroadcast:Boolean
hasScreenPlayback:Boolean
hasSharedObjects:Boolean
hasSMAF:Boolean
hasSMS:Number
hasStreamingAudio:Boolean
hasStreamingVideo:Boolean
hasStylus:Boolean
hasVideoEncoder:Boolean
hasXMLSocket:Number
imageMIMETypes:Array
isDebugger:Boolean
language:String
localFileReadDisable:Boolean
MIMETypes:Array
os:String
screenOrientation:String
screenResolutionX:Number
screenResolutionY:Number
softKeyCount:Number
version:String
videoMIMETypes:Array
```

www.it-ebooks.info

SYSTEM.SECURITY

METHODS

```
allowDomain(domain:String):Void
allowInsecureDomain(domain:String):Void
loadPolicyFile(url:String):Void
```

TEXTFIELD

PROPERTIES

```
_alpha:Number
autoSize:Object
background:Boolean
backgroundColor:Number
border:Boolean
borderColor:Number
bottomScroll:Number
condenseWhite:Boolean
embedFonts:Boolean
_height:Number
_highquality:Number
hscroll:Number
html:Boolean
htmlText:String
length:Number
maxChars:Number
maxhscroll:Number
maxscroll:Number
multiline:Boolean
_name:String
_parent:MovieClip
password:Boolean
_quality:String
_rotation:Number
scroll:Number
selectable:Boolean
_soundbuftime:Number
tabEnabled:Boolean
tabIndex:Number
_target:String
text:String
```

www.it-ebooks.info

PROPERTIES

```
textColor:Number
textHeight:Number
textWidth:Number
type:String
_url:String
variable:String
_visible:Boolean
_width:Number
wordWrap:Boolean
_x:Number
_xmouse:Number
_xscale:Number
_y:Number
_ymouse:Number
_yscale:Number
```

METHODS

```
addListener(listener:Object):Boolean
getDepth():Number
getNewTextFormat():TextFormat
removeListener(listener:Object):Void
removeTextField():Void
replaceSel
replaceText(beginIndex:Number, endIndex:Number, newText:String):Void
setNewTextFormat(tf:TextFormat):Void
setTextFormat([beginIndex:Number],
                [endIndex:Number], textFormat:TextFormat):Void
```

EVENTS

```
onChanged = function(changedField:TextField) {}
onKillFocus = function(newFocus:Object) {}
onScroller = function(scrolledField:TextField) {}
onSetFocus = function(oldFocus:Object) {}
```

www.it-ebooks.info

TEXTFORMAT

CONSTRUCTOR

```
TextFormat([font:String],[size:Number],[color:Number],[bold:Boolean],
[italic:Boolean],[underline:Boolean],[url:String], [target:String],[align:
String],[leftMargin:Number],
[rightMargin:Number],[indent:Number],[leading:Number])
```

PROPERTIES

```
align:String
blockIndent:Number
bold:Boolean
bullet:Boolean
color:Number
font:String
indent:Number
italic:Boolean
leading:Number
leftMargin:Number
rightMargin:Number
size:Number
tabStops:Array
target:String
underline:Boolean
url:String
```

METHODS

```
getTextExtent(text:String, [width:Number]):Object
```

VIDEO

METHODS

```
attachVideo(source:Object):Void
close():Void
pause():Void
play():Boolean
resume():Void
stop():Void
```

www.it-ebooks.info

EVENTS

```
onStatus = function(infoObject:Object) {}
```

XML

CONSTRUCTOR

```
XML(text:String)
```

PROPERTIES

```
contentType:String
docTypeDecl:String
ignoreWhite:Boolean
loaded:Boolean
status:String
xmlDecl:String
```

METHODS

```
addRequestHeader(header:Object, headerValue:String):Void
createElement(name:String):XMLNode
createTextNode(value:String):XMLNode
getBytesLoaded():Number
getBytesTotal():Number
load(url:String):Boolean
parseXML(value:String):Void
send(url:String, [target:String], method:String):Boolean
sendAndLoad(url:String, resultXML:XML):Void
toString():String
```

EVENTS

```
onData = function(src:String) {}
onLoad = function(success:Boolean) {}
```

www.it-ebooks.info

XMLNODE

PROPERTIES

```
attributes:Object
childNodes:Array
firstChild:XMLNode
lastChildXMLNode
nextSibling:XMLNode
nodeName:String
nodeType:Number
nodeValue:String
parentNodeXMLNode
previousSibling:XMLNode
```

METHODS

```
appendChild(newChild:XMLNode):Void
cloneNode(deep:Boolean):XMLNode
hasChildNodes():Boolean
insertBefore(newChild:XMLNode, insertPoint:XMLNode):Void
removeNode():Void
toString():String
```

XMLSOCKET

CONSTRUCTOR

```
XMLSocket()
```

METHODS

```
close():Void
connect(url:String, port:Number):Boolean
send(data:Object):Void
```

EVENTS

```
onClose = function() {}
onConnect = function(success:Boolean)
onData = function(src:String) {}
onXML = function(src:XML) {}
```

www.it-ebooks.info

INDEX

A

Accelerometer, 440, 441–447
account requests, 289–290
ActionScript (AS)
  1.1/2.0 Engine, 8
  classes (list), 509–536
ActionScript 2.0 (AS2), 9, 33–35
ActionScript 3 (AS3), 9, 33, 107, 109
`activateControls()`, 233
actors, 109, 126, 137. *See also* commands; mediators; proxies; PureMVC
Adaptive Differential PCM (ADPCM), 10
`Add()`, 36, 348, 349, 353, 356, 357, 358, 360, 363
`addBroadcastMessageReceivedListener()`, 470
`addButton()`, 80, 81
`addCheckBox()`, 85, 86
`addClientDisconnectedListener()`, 448
`addContact()`, 458, 459, 460
`addDataReceivedListener()`, 441, 444
`addEvent()`, 454
`addEventListener()`, 64, 67, 272, 409, 417
`addFileSystemChangedListener()`, 461
`addItems()`, 95, 453
`addMessageReceivedListener()`, 448
`addMMSReceivedListener()`, 477
`addOrientationChangedListener()`, 441, 444
`AddRadioBtn()`, 88, 89
`addRequestHeader()`, 521, 535
`addSMSReceivedListener()`, 470
`addSubCommand()`, 117, 118, 160, 220, 282, 317, 319, 423
Adobe Device Central CS4. *See* Device Central
Adobe Flash CS4 Professional. *See* Flash CS4
Adobe Flash Lite. *See* Flash Lite 3.*x*
Adobe Generic Phone device template, 16, 26
ADPCM (Adaptive Differential PCM), 10
aggregators, 3–4, 501
`Aino` class, 36–38
  class name, package name, constructor (listing 3–1), 36
  extending Mobile and calling its constructor (listing 3–8), 44
  getter/setter methods (listing 3–4), 37–38
  instance (listing 3–3), 37
  manufacturer/model properties (listing 3–2), 37
  owner (variable), 44
  setting properties for instance (listing 3–5), 38
Aino/Nokia E71 (example), 35–47
`allowDomain`, 521, 532
`allowInsecureDomain`, 521, 532
Anniversary entry, 350
APIs. *See* Google Map API; Google Weather API; Project Capuchin Platform Service APIs; S60 Platform Services APIs
app stores, 4
application icons, 501–502
application signing, 489, 496–501
  Java Platform, 500–501
  operating systems and, 496
  S60 Platform, 500
  self-signing, 500
  Symbian Signed program, 489, 500, 501
  Windows Mobile platform, 500
`ApplicationFacade`, 111–114
  creating (listings 5–1, 5–20, 5–21), 112, 127–128
  declaring notification names (listing 5–4), 114–115
  Image Viewer client, 256–258
  Media Console, 193–196
  notification names in
    Image Viewer, 256–257
    Media Console, 194–195
    TV Listings application, 144
    Weather Client, 399–400
  overriding `getInstance()` (listing 5–2), 112–113
  overriding `initializeController()` (listing 5–3), 113–114
  startup method (listing 5–5), 116
  TV Listings application (listing 6–1), 143–144
  Twitter client, 299–300
  Weather Client application, 399–400
`apply()`, 518
AppManager (S60 Platform Services API), 339, 343–347
  apps
    distinguishing, 344
    filtering information on, 344
    retrieving information on, 343–344
  `GetList()` and, 344–346

www.it-ebooks.info

AppManager (*continued*)
launching applications/documents with, 346–347
`AppMediator`
creating
listings 5–14 to 5–18, 122–124
listings 5–32 to 5–35, 134–135
`handleNotification()` for, 121–122, 124
`listNotificationInterests()` for, 121, 124
Arial font, 399
`Array()`, 85, 200, 518
`Array`
constants, 510
constructor, 509
methods, 510
properties, 509
AS. *See* ActionScript
asynchronous methods, 340–343
`attachAudio()`, 189, 236
attachments, 372–373
`attachMovie()`, 80, 86, 89, 123
`attachSound()`, 185, 215, 217
audio playlist, 197–198
audio support, 10
`audioMIMETypes`, 63, 531
AUDIO_STREAM_CREATED, 235
`AuthenticateUserCommand`, 318–319
authentication (Twitter client), 318–319
Automated Testing section, 26

B

`backgroundColor`, 81, 83, 100, 101, 105, 327, 532
BACKSPACE, 69, 520
Balkan, Aran, 285, 292, 336. *See also* SWX format
`Battery` class (listing 3–12), 49–50
BBC iPlayer, 139
bitmaps, 243. *See also* `Flash.Display.BitmapData`
Device Profile panel and, 17
Flash Lite architecture and, 8
`Mifconv` tool and, 502
block request, 290
blocking, following and, 287
blueprint, 34, 38. *See also* class diagram
Bluetooth (Project Capuchin Service API), 440, 448–454
`Boolean()`, 518
`Boolean`
constructor, 510
methods, 510
broadcaster, 140
Browne, Pedr, 107
`bufferLength`, 181, 526
`bufferTime`, 181, 211, 526
`Button` class
events, 511
method, 511
properties, 511
buttons, 72–74, 77–81
benefits, 78
push buttons, 78–81
states, 78
using `PushButton` (listing 4–1), 79–81
radio, 87–90
lists with, 95–96

C

CAB files, 486, 497, 500
Cabinet (CAB) files, 486, 497, 500
`Calculate()`, 366, 368, 369
Calendar (Project Capuchin Service API), 440, 454–457
Calendar (S60 Platform Services API), 339, 347–353
calendar entries
creating, 349–350
exporting, 351–352
importing, 352–353
Calibrate (Device Performance panel), 29–30
`call()`, 296, 518
`Cancel()`, 341, 343, 348, 360, 370, 378
`cancelDevicesSearch()`, 448
`cancelNotification()`, 366, 368, 370, 377
capabilities (mobile devices), 10, 60–64, 531. *See also* `System.capabilities`
communication features, 64
external data support, 62
information about system, 61–62
key input/device interaction, 62–63
media capabilities, 63–64
CAPSLOCK, 69, 520
carriers, 3, 16, 17. *See also* network operators
certificates, 497, 498–499
Certify (Sony Ericsson submissions procedure), 507
`ChangeStatus()`, 370, 377
check boxes, 84–86, 95
`CheckBox` UI component (listing 4–4), 85–86
`checkTime()`, 214
Chumby, 7
`class` (keyword), 34
class diagram (Aino/Nokia E71 example), 38–47
classes (ActionScript 2.0), 509–536. See also *specific classes*
Image Viewer client, 244–255
media, 179–192
`clearInterval()`, 65–66, 213, 518
`close()`, 179, 180, 209, 210, 213, 441, 445, 521, 526, 534, 536
`closeConnection()` (listing 7–27), 210
CMIDI sound, 63, 531
ColdFusion, 109
`Color`
constructor, 512
methods, 512
Command (design pattern), 111, 116–119
commands, 116–119. *See also specific commands*
Controller and, 116–119
registering, 119, 128
communication features (`System.capabilities`), 64
`compareStrings()`, 465
Component Inspector, 78–81
compound sound data, 63
conceptual diagrams, 109, 126

www.it-ebooks.info

Image Viewer
controller, 276–277
model, 258–259
view, 269
Media Console
controller, 219
model, 196–197
view, 225
PureMVC, 110
PureMVC example application, 126–127
TV Listings application
controller, 159
model, 145
view, 166
Twitter client
controller, 316
model, 301
view, 323
Weather Client application
controller, 422
model, 401
view, 427
`ConditionVO`, 402
Confirmations (dialogue component), 96, 97
`connect()`, 179, 190, 526
`connectToServer()`, 448
contact key values, 356
Contacts (Project Capuchin Service API), 440, 457–461
Contacts (S60 Platform Services API), 337, 353–360
contacts/databases/groups
adding, 357–358
organizing, 359–360
retrieving, 354–355
`GetList()` and, 355–357
methods, 353
vCard files and, 358–359
content (Flash Lite content)
aggregators, 3–4, 501
content producers, 3
meta data, NetStream and, 182–183
packaging, 485–508
testing, 30–31
types, 7–8
emulator and, 26
untrusted, 497–498
content submission (Sony Ericsson), 507–508
Content Type section, 26
continuous play option, 85
CONTROL, 69, 520
Controller. *See also* PureMVC
commands and, 116–119
Image Viewer client, 275–282
Media Console, 218–224
in MVC, 109
PureMVC example application, 132–134
TV Listings application, 158–165
Twitter client, 316–322
Weather Client application, 421–426
Core Project Capuchin MXP Service, 441. *See also* Project Capuchin Platform Service APIs
Core Rendering Engine, 8
`createData()` (listing 5–26), 130–131, 132
`createDirectory()`, 461
`createFile()`, 462
`createServer()`, 448
`crossdomain.xml` file, 176
CSV output (Google Map API), 391
Current Weather Screen (wireframe), 397
`CurrentConditionsMediator`, 428–430

D

databases/groups. *See* Contacts
`Date`
constructor, 512
methods, 512–513
`dayHelper()`, 404, 405, 406, 408
decode, 395
`decode()`, 521
`Delete()`, 348, 353, 360, 370, 378
DELETEKEY, 69, 520
deleting messages, 37
Deployment (Sony Ericsson submissions procedure), 507
design, wireframes and, 397–398
design patterns, 109–111
Command, 111, 116–119
Facade, 110, 111–114
Factory, 110
Mediator, 111, 119–124
Observer, 110, 114–116
Proxy, 111, 125–126
Singleton, 110, 112, 115, 120
`DetailsMediator` (listing 6–27), 165, 172–173
development frameworks. *See also* PureMVC
need for, 107–109
PureMVC, 107–137
Device Central (Adobe Device Central CS4), 14–31
Accelerometer and, 446–447
advantages, 15
content types and, 8
device profiles, 16–17
comparing, 18–20
list of, 22–23
device sets, 15–18
emulator. *See* emulator
features of, 14–15
“Hello World” example, 23–26
Image Viewer in, 283
local library, 16
Media Console application in, 238
online library, 16
PureMVC example application in, 136
requirements for, 14
TV Listings application in, 174–175
Twitter client in, 336
Weather Client application in, 436
device clock, 417–420
Device Performance panel, 27, 28, 29–30
device profiles, 16–17. *See also* Nokia 5800 XpressMusic; Nokia E71; Nokia N95 8GB; Sony Ericsson
comparing, 18–20
display sizes, 19–20
list of, 22–23

www.it-ebooks.info

Device Profiles panel, 16, 18, 19, 20
device sets, 15–18
  creating, 17–18
  defined, 15–16
Device Status panel, 27, 31, 50, 84
DeviceAnywhere, 31
diagrams. *See* class diagram; conceptual diagrams
`Dialogue` UI component (listing 4–17), 96–97
direct message request, 290
direct messaging, 287
`Disabled` (UI component state), 78
Disabled and Highlighted, 78
`disconnectAllClients()`, 448
`disconnectFromServer()`, 448
`dispatchEvent()`, 64, 66
  `removeListener()` and, 247
  `tickComplete()` and, 66–67
Display section, 26
display sizes (device profiles), 19–20
DISPLAY_RESULTS, 144, 145, 152, 166, 170, 171, 172, 173, 174
documents (Flash mobile documents)
  creating, 21–23
  File ⇒ New Document In ⇒ Flash path, 20, 21
DOWN (static property), 69, 520
dual row lists, 94–95
`DuplicateMovieClip()`, 518, 524
`dynamic` (keyword), 34
dynamic data, 9, 12
dynamic heap value, 29, 54, 55
dynamic text, 77

E

`E71` class, 45–47
  class diagram and, 38, 39
  overriding switchOn/playRingtone methods (listings 3–10, 3–11), 46–47
ECMA (European Computer Manufacturers Association), 34
ECMA-262 standard, 9, 34
ECMAScript Language Specification, 34
`_effectColor`, 99, 100
embedded video, 63
emulator, 26–30
  Automated Testing section, 26
  Content Type section, 26
  Device Performance panel, 27, 28, 29–30
  Device Status panel, 27, 31, 50, 84
  Display section, 26
  File Info section, 26
  Key Pad section, 26
  Memory panel, 26, 27–29
  Message section, 27
  Network Performance, 27
  Network Status, 27, 84
  Persistent Storage, 27
  Security panel, 27, 176
`Enabled` (UI component state), 78
encapsulation, 35, 111
END, 69, 520
ENTER, 69, 520
`epocrc.exe` tool, 503–505
`Error` class
  constructor, 513
  method, 513
  properties, 513
error codes (S60 Platform Services APIs), 342
Errors (dialogue component), 96, 97
ESCAPE, 69, 520
`escape()`, 519
European Computer Manufacturers Association. *See* ECMA
`eval()`, 519
event listeners, 64–68
`EventDispatcher` class, 64–68
events, 64–68
`EventsDispatcher.initialize()`, 65, 382
`execute()`, 117, 122, 125, 132, 163, 222, 223, 271, 278, 280, 318, 319, 320, 322, 423, 425, 426
exporting
  calendar entries, 351–352
  Landmarks, 364–365
  vCard files, 359
`ExtendBackLightDuration`, 60
`ExtendedKey`, 71–72, 514
extending `MacroCommand` (listing 5–7), 118
`extends` (keyword), 34
external data support (`System.capabilities`), 62
external .mp3 file (listing 7–1), 186

F

Facade (design pattern), 110, 111–114. *See also* ApplicationFacade
Factory (design pattern), 110
`faultHandler()`, 313
favorites request, 291
feature phones, 2
File (Project Capuchin Service API), 440, 461–465
File Info section, 26
File ⇒ New Document In ⇒ Flash path, 20, 21
`FileTypes` class, 379
`Filter` object property, 342, 343
filtering messages, 374
`FilterVO` class, 379, 380
`.fla` file, 22
  Hello World, 24
  Image Viewer client, 256
  Media Console, 192–193
  `Memory` class and, 54
  `NetworkStatus` class and, 56
  sizes for, 22
  TV Listings application, 143
  Twitter client, 299
  Weather Client application, 399
Flash CS4 (Adobe Flash CS4 Professional), 14
  document creation with, 21–23
  New Document panel, 20, 21–22, 23
  requirements for, 14
  trial version, 15
Flash enabled, 6
Flash enabled mobile device applications
  Image Viewer client, 241–283, 491–495
  Media Console, 179–239

www.it-ebooks.info

PureMVC example, 126–136
TV Listings application, 139–177
Twitter client, 285–336
Weather Client application, 385–437
Flash Lite 3.*x*
architecture, 8–9
content. *See* content
features, 9
history, 5–6
JME and, 7
versions, 5–6
Flash Lite mobile application development
OEMs' support and, 7
tools for, 13–31
Flash Media Server (FMS), 6, 10, 11, 179, 190, 191, 239
Flash Player 10.1, 6
Flash section (Device Profile panel), 17
Flash Video (FLV), 10–11
`Flash.Display.BitmapData`
constructor, 514
methods, 514–515
properties, 515
`Flash.Geom.ColorTransform`, 515
`Flash.Geom.Matrix`, 516
`Flash.Geom.Point`, 516–517
`Flash.Geom.Rectangle`, 517–518
Flickr, 297. *See also* Twitter
`flush()`, 255, 268, 528
FMS (Flash Media Server), 6, 10, 11, 179, 190, 191, 239
following, blocking and, 287
font support, 11
football score, 104
`ForecastConditionsMediator`, 430–433
`forecast_Information`, 387, 388
`formatCurrency()`, 465
`formatDateTime()`, 465
`formatMessage()`, 465
`formatNumber()`, 465
`formatPercentage()`, 465
Formula One team Brawn GP, 241
Forum Nokia
"Guide to Flash Lite Components," 105
RDA and, 31
UI components, 76–78, 105
4 Day Weather Forecast Screen (wireframe), 397
friendships request, 290–291
FRIENDS_TIMELINE, 310–311
`FriendsTimelineMediator`, 334–336
`fscommand()`, 10, 47–48, 519
`fscommand2()`, 10, 21, 47, 48–60, 519
`Battery` class (listing 3–12), 49–50
battery status and, 48–50
device vibration and, 59
device volume and, 59
memory allocation and, 52–55
network requests/connections and, 55–59
network signal and, 51–52
online information, 60
scenarios for using, 59–60
fscommands, 10, 47–48
`FSCommandsProxy`
TV Listings application, 144, 145, 146
Weather Client, 420–421
`FullScreen`, 59
`Function`, 518

G

General section (Device Profile panel), 17
Generic Phone device template, 16, 26
generic properties/features (mobile devices), 35–36
geocoding, 390
GET, 288
`get` (keyword), 34
`getArtist()`, 199, 202, 203, 206
`getAscii()`, 520
`GetBatteryLevel`, 48
`getBluetoothAddress()`, 448
`getBytesLoaded()`, 186, 521, 524, 529, 535
`getBytesTotal()`, 186, 521, 524, 529, 535
`getComparingLocales()`, 466
`GetDevice`, 60
`GetDeviceID`, 60
`getDeviceName()`, 448
`getDuration()`, 199, 202, 203, 206
`getEntries()`, 458
`getEvents()`, 454
`getFile()`, 199, 202, 203, 206, 207
`getFileInfo()`, 462
`getFormattingLocales()`, 466
`GetFreePlayerMemory`, 53
`getFrequency()`, 475
`GetGoogleWeatherCommand`, 424
`GetHandsetLocationCommand`, 425
`getImagesFromFriends Timeline()`, 313–314
`getInstance()`, 112–113, 194, 195, 256, 399
`GetJar`, 501
`getJsonObject()`, 395, 396, 397, 417
`GetList()`
AppManager and, 344–346
Calendar and, 347
Contacts and, 355–357
Media Management and, 378
Messaging and, 370, 372–374
`getLocal()`, 255, 267, 268, 528
`getLocale()`, 466
`getLocation()`, 365, 366, 368, 413, 414, 416, 425
`GetMapLocationCommand`, 425
`GetMaxBatteryLevel`, 48
`getMaxFreq()`, 475
`GetMaxVolumeLevel`, 59
`getMediatorName()`, 121, 237, 431
`getMiniFreq()`, 475
`getModulation()`, 475
`getNetConnection()`, 214
`getNetStream()`, 214
`GetNetworkConnectStatus`, 56
`GetNetworkRequestStatus`, 56
`GetNetworkStatus`, 55
`getNextHighestDepth()`, 80
`GetNextImageCommand`, 279
GET_NEXT_RESULT, 144, 164, 166, 170, 171, 175

www.it-ebooks.info

GetNextResultCommand (listing 6–21), 164
GetNextSectionCommand, 126, 128, 132, 133
getNumberOfConnected Clients(), 448
getNumberOfPresets(), 476
getOrientation(), 441, 445, 446, 447
getPan(), 189, 529
getParsedData(), 408, 411
GetPlatform, 60
GetPowerSource, 48
GetPreviousImageCommand, 279
GET_PREVIOUS_RESULT, 144, 164, 166, 170, 171, 175
GetPreviousResult (listing 6–22), 164
GetPreviousSectionCommand, 126, 128, 132, 133–134
getProgress(), 98, 244, 525
getRDSData(), 476
getScreenNamesFrom FriendsTimeline(), 314
getSelected(), 154
getServer(), 199, 202, 203, 206, 207
getSize(), 255, 268, 528
getSoundMode(), 476
getStatusesFromUser Timeline(), 314
GetStoredImageCommand, 280–281
getTimer(), 519
getTitle(), 202, 203, 206
GetTotalPlayerMemory, 53
getTransform(), 189, 512, 529
getUserAt(), 314
getValues(), 441, 445, 446
getView(), 454
getVolume(), 189, 218, 476, 529
GetVolumeLevel, 59
getVolumeLevel(), 476
getWidth(), 99, 100, 104, 248, 249
Global, 518–519
Google API key, 390
Google Map API, 390–397
  CSV output, 391
  JSON output, 392–393
  parsing JSON, 393–397
  XML/KML output, 391–392
Google Weather API, 386–390
GoogleMapProxy, 401, 414–417
GoogleWeatherLoader, 403–404
GoogleWeatherParser, 404–408
GoogleWeatherProxy, 401, 409–411
groups/databases. *See* Contacts
"Guide to Flash Lite Components" (Forum Nokia), 105

H

H.264, 11
Hall, Cliff, 107
Handango, 501
handleNotification()
  for AppMediator, 121–122, 124
  for MediaMediator, 232
  for PlaylistMediator, 227–228
has4WayKeyAS, 62, 531
hasAccelerometer, 40, 41, 42, 44, 45, 46
hasAudio, 63, 192, 531
hasCMIDI, 63, 531
hasCompoundSound, 63, 531
hasDataLoading, 62, 531
hasEmail, 41, 43, 64, 531
hasEmbeddedVideo, 63, 192, 531
hasGPS, 41, 43
hash tags (Twitter), 287
hasInternet, 41, 42
hasMappableSoftKeys, 62, 531
hasMFI, 63, 531
hasMIDI, 63, 531
hasMMS, 64
hasMouse, 63
hasMP3, 63, 192, 531
hasQWERTYKeyboard, 62, 531
hasQWERTYKeys, 40, 41, 42, 44, 45, 46
hasSharedObjects, 62, 531
hasSMAF, 63, 531
hasSMS, 64, 531
hasStreamingAudio, 63, 192, 531
hasStreamingVideo, 63, 192, 531
hasStylus, 63, 72, 531
hasTouchUI, 40, 41, 42, 44, 45, 46
hasXMLSocket, 62, 531
haXe, 109
HD (high definition), 11
heap values, 28–29, 54–55
  dynamic, 29, 54, 55
  static, 28– 29, 54, 55
"Hello World" example, 23–26
hideComponents, 233
high definition (HD), 11
Highlighted (UI component state), 78
HOME, 69, 520
Home Screen (wireframe), 397
HTC, 7

I

I18n (Project Capuchin Service API), 440, 465–469
iCalendar, 348–349
icons
  application (Nokia N97), 501–502
  lists with, 94–95
  weather, 389–390
ID3 information file, 187–188
Image class, 244–249
image grid, 281
Image Viewer client, 241–283
  ApplicationFacade, 256–258
  classes in, 244–255
  conceptual diagrams
    controller, 276–277
    model, 258–259
    view, 269
  controller, 275–282
  in Device Central, 283
  .fla file, 256
  imageViewer.pkg file for, 491–495
  model, 258–268
  Nokia E71, 283
  view/mediators, 268–275
ImageGrid class, 249–254
ImageGridMediator, 271–274
imageMIMETypes, 63, 242, 531
imagePathHelper(), 404, 405, 406, 407, 408
ImageProxy, 263–266
images, 11
  challenges, 241–242
  collections, 263–266

www.it-ebooks.info

memory considerations, 243
multiple, loading, 242–243
navigating through, 279
sizes, 243
supported, 242
ImagesXMLProxy, 260–263
imageViewer.pkg file, 491–495
imageViewer_reg.rsc, 503
imageViewer.rss, 503
ImageVO, 259–260
imei, 41, 43, 60
IMobile interface
class diagram and, 38, 39
listing 3–6, 40
public methods, 40
implements (keyword), 34
Import(), 348, 352, 353, 358, 360, 364
import (keyword), 34
importing
calendar entries, 352–353
Landmarks, 364–365
ModelPrepCommand and ViewPrepCommand (listing 5–8), 118
vCard files, 358–359
index value (Device Performance panel), 30
indicators (visual indicators), 98–100
Information (dialogue component), 96, 97
information about system (System.capabilities), 61–62
inheritance, 44, 112, 115, 220, 244, 246, 249
init(), 441
listing 3–20, 65, 66
listing 7–26, 210
initCall(), 307–308
initialize(), 65, 403, 413, 414, 425
initializeController(), 113–114, 128, 193, 195, 196, 256, 257, 300, 399, 400
initializeFacade(), 113
initializeMacroCommand(), 117, 118, 160, 220, 282, 317, 318, 423
initializeModel(), 112
initializeView(), 112, 113
initInterval(), 214
INIT_SOUND, 193, 195, 196, 210
initTimer(), 157, 163
_inlinePicture property, 98
input text, 77
INSERT, 69, 520
interaction/navigation (mobile devices), 4
interface (keyword), 34
internationalizing currency, 466–468
iPhones, 4
iRiver, 7
_isBackgroundVisible, 83, 84, 101
isBluetooth(), 448
isDayorNight(), 420
isDialogueVisible, 96, 97, 98
isMP3(), 202, 203, 206, 207
isMuted(), 476
isPlaying(), 476
isRDSSignal(), 476
isRTMP(), 202, 203, 206, 207
iTunes, 179

J

JAR files, 486, 487–488, 497, 500, 501, 507
Java Platform
Micro Edition, 7
packaging and, 486
signing for, 500–501
Java Runtime Environment (JRE), 487
JPEG, 9
JRE (Java Runtime Environment), 487
JSON
output (Google Map API), 392–393
parsing, 393–397
Twitter client and, 288, 295

K

Kemp, Eric, 188
Key class, 68–71
events, 520
methods, 520
static properties, 69, 520
key input/device interaction (System.capabilities), 62–63
key listeners, 68, 70, 71, 72, 135, 170, 172, 272, 442, 476
Key Pad section, 26
Key.addListener() (listing 3–23), 68, 70, 135, 246, 255, 520
Key.getCode(), 69, 70, 520
Key.isDown(), 520
keypads/keyboards, 68
QWERTY, 36, 38, 40, 60, 62, 71
hasQWERTYKeyboard, 62, 531
hasQWERTYKeys, 40, 41, 42, 44, 45, 46
Key.removeListener(), 70–71, 520
keys, 11. *See also* soft keys
Keys class, 380–381
keywords, 34–35
KML/XML output (Google Map API), 391–392
Kuala Lumpur, server response for, 387–388

L

Landmarks (S60 Platform Services API), 339, 360–365
categories, 365
creating, 361–364
information in, 361, 363
methods, 360
Language (imageViewer.pkg), 492
language (System.capabilities), 59, 61, 62
language parameter (Google Weather API), 386–387
LaunchApp(), 343, 346, 347
LaunchDoc(), 343, 346, 347
LEFT, 69, 520
Linkage Identifier, 80, 86, 185, 189

www.it-ebooks.info

List UI component, 90–96
  dual row lists, 94–95
  single row lists (listings 4–6 to 4–14), 90–94
_listeners:Array, 520
listFiles(), 462
listFileSystems(), 462
listNotification(), 227–228
listNotification Interests()
  for AppMediator, 121, 124
  for MediaMediator, 231
listPresets(), 476
ListRadioButton UI component, 95–96
lists, 90–96
  with check boxes, 95
  dual row, 94–95
  with icons, 94–95
  with radio buttons, 95–96
  single row (listings 4–6 to 4–14), 90–94
ListTwoRowWithIcon UI component (listing 4–15), 94–95
LoadImagesXMLCommand, 278
loading external resources, 10
loadMovie(), 62, 63, 64, 102, 176, 244, 389, 519, 524
loadPlaylist(), 20, 199, 203, 204, 222
LOAD_PLAYLIST, 193, 194, 196, 225, 227
loadPolicyFile, 532
loadSound(), 62, 185, 186, 215, 217, 529
loadVars, 62, 176, 393, 394, 520–521
  constructor, 520
  events, 521
  methods, 521
  properties, 521
local library, 15, 16, 18, 22
LocalConnection, 521
location (Google Weather API), 386
Location (Project Capuchin Service API), 440, 483
Location (S60 Platform Services API), 339, 365–369
  measuring distances, 368–369
  methods, 365–366
  retrieving locations, 366
    automatic, 367–368
    canceling calls and, 368
  Weather Client and, 385
LocationProxy, 401, 411–414
Logging (S60 Platform Services API), 339, 384
LoginProxy, 315

M

Mac OS X, 14, 338
MacroCommand, 116–117
  extending (listing 5–7), 118
  initializeMacroCommand(), 117, 118, 160, 220, 282, 317, 318, 423
Macromedia. *See also* Flash Lite
  Flash Lite and, 5
  Flash MX, 34, 255
  MAX 2005, 1
MainMenuMediator (listing 6–24), 165, 167–168
maintenance headaches, 108
makeCall(), 40
makeConnection() (listing 7–28), 210–211
MakeKeys, 489, 490, 499
makekeys.sis, 498–499
makesis.exe tool, 489, 495, 496, 498
makeStream() (listing 7–29), 211
mandatory search parameters, 140
manufacturer property, 38, 41, 43
Math, 522
Math.floor(), 181
MAX 2005, 1
measuring distances, 368–369
media capabilities (System.capabilities), 63–64
media classes, 179–182
Media Console mobile application, 179–239
  ApplicationFacade, 193–196
  conceptual diagrams
    controller, 219
    model, 196–197
    view, 225
  controller, 218–224
  in Device Central, 238
  .fla file, 192–193
  model, 196–218
  Nokia N95 8GB and, 238
  view/mediators, 224–238
Media Management (S60 Platform Services API), 339, 378–383
  methods, 378
  retrieving meta data, 378–383
media owners, 3
MediaManagement class, 382–383
MediaMediator (listings 7–60 to 7–72), 229–238
MediaProxy, 197, 206
  creating (listing 7–24), 208–209
  init() (listing 7–26), 210
  methods, 207–208
  private variables (listing 7–25), 209
MEDIA_TIME_UPDATE, 236
Mediator (design pattern), 111, 119–124
mediators
  Image Viewer, 268–275
  Media Console, 224–238
  registering, with application, 120–122
  TV Listings application, 165–174
  Twitter client, 322–336
  View and, 119–124
  Weather Client application, 426–436
Melody Format (MFi), 63
memory (mobile devices)
  fscommand2()and, 52–55
  heap values, 28–29, 54–55
    dynamic, 29, 54, 55
    static, 28– 29, 54, 55
  images and, 243
  issues, 4, 243
Memory class (listing 3–14), 53–54
Memory panel, 26, 27–29
mentions (Twitter), 288
Message section, 27
Messaging (Project Capuchin Service API), 440, 470–471
Messaging (S60 Platform Services API), 339, 369–378

www.it-ebooks.info

messages, 370–371
attachments, 372–373
deleting, 37
filtering, 374
modifying status, 377–378
monitoring inbox, 376–377
retrieving, 371–372
sending, 375–376
sorting, 374–375
methods, 370
meta data
NetStream and, 182–183
onID3() and, 188
onMetaData and, 182, 207, 209, 211, 212, 217, 526
retrieving, 180, 378–383
metaInfo object, 183
methods. *See also specific methods*
ActionScript 2.0, 509–536
S60 Platform Services APIs, 338–343
asynchronous, 340–343
synchronous, 339–340
MFi (Melody Format), 63
MIDI audio format, 10, 63, 483
hasMIDI, 63, 531
MIDlet, 487
MIDP (Mobile Information Device Profile), 487
mifconv.exe, 502–503
MIME types, 63
audioMIMETypes, 63, 531
imageMIMETypes, 63, 242, 531
videoMIMETypes, 64, 192, 531
MIMETypes (System.capabilities), 64, 344, 364, 365, 376, 381, 531
MMS, 61, 64
hasMMS and, 64
MessageType and, 371
messaging (Project Capuchin Service APIs), 440, 470–471
Ovi account and, 506
Sending MMS from within an application (listing 10–40), 376
Sending to multiple recipients (listing 10–41), 376
sorting messages retrieved from an inbox (listing 10–38), 375
Mobile (base class)
class diagram and, 38, 39
extending Mobile and calling its constructor (listing 3–8), 44
private properties, 40–41
properties/methods defined (listing 3–7), 41–44
updating manufacturer/model setter methods (listing 3–9), 45
mobile application development. *See* Flash Lite mobile application development
mobile devices, 1–2
applications
Image Viewer client, 241–283, 491–495
Media Console, 179–239
PureMVC example, 126–136
TV Listings, 139–177
Twitter client, 285–336
Weather Client, 385–437
capabilities (System.capabilities), 60–64. *See also* System.capabilities
communication features, 64
external data support, 62
information about system, 61–62
key input/device interaction, 62–63
media capabilities, 63–64
feature phones, 2
generic properties/features, 35–36
issues
interaction/navigation, 4
memory, 4
processing power, 4
screen size, 4
memory. *See* memory
native properties, 47
smartphones, 2, 4, 5, 337
testing content on, 30–31
mobile ecosystem, 2–4
Mobile Information Device Profile (MIDP), 487
Mobile TV Guide, 139–140
mobileNum property, 41, 43, 44, 45, 46, 47
modal dialogues, 96–98
Model
Image Viewer client, 258–268
Media Console, 196–218
in MVC, 109
proxies and, 125–126
PureMVC example application, 128–132
TV Listings application, 144–158
Twitter client, 300–315
Weather Client application, 400–421
model (property), 38, 41, 43
ModelPrepCommand, 117, 118, 125, 126, 132, 133, 219, 220, 268, 276, 277, 316, 317, 422, 423
Image Viewer (listing 8–53), 277
Media Console (listing 7–45), 221
TV Listings application (listing 6–15), 160–161
Twitter client, 317
Weather Client, 423
Model-View-Controller (MVC), 109. *See also* Controller; Model; PureMVC; View
monitoring inbox, 376–377
Mouse, 522–523
events, 523
methods, 522
MovieClip, 523–525
events, 525
methods, 524
properties, 523
MovieClipLoader, 243, 525
events, 525
Image Viewer client, 244–254
methods, 525
MP3, 10, 61, 63
hasMP3, 63, 192, 531
mobile MP3 player application, 85–86
mp3Settings, 95
Multimedia (Project Capuchin Service API), 440, 483
MVC (Model-View-Controller), 109. *See also* Controller; Model; PureMVC; View

www.it-ebooks.info

N

Nabaztag, 297
native properties, 47
navigation, 11
NetConnection, 179–180
  constructor, 526
  methods, 179, 526
NetStream, 180–184
  attributes, 180
  buffer monitoring, 181
  constructor, 526
  content meta data, 182–183
  data load, 182
  events, 526
  methods, 526
  playback handling, 180
  playback progress, 181–182
  properties, 526
network operators, 3, 16, 17, 31, 496, 500, 509
Network Performance, 27
network requests/connections, fscommand2() and, 55–59
network signal, fscommand() and, 51–52
Network Status Panels, 27, 84
NetworkConnection class
  listing 3–17, 58–59
  listing 6–11, 155–156
NetworkMediator (listing 6–28), 165, 173–174
NetworkProxy, 144, 145, 155–158
NetworkRequest class (listing 3–16), 56–58
NetworkStatus class (listing 3–15), 55–56
NETWORK_STATUS_UPDATED, 145, 158, 166, 173, 174
New Document Panel, 20, 21, 22, 23
nextResult(), 154
nextSection() (listing5–27), 131–132, 133
Nokia. *See also* S60 Platform Services APIs
  6210 model, 493
  Flash mobile development and, 7
  Flash Packaging Tool, 486, 505–506
  Ovi, 506–507
Nokia 5800 XpressMusic, 23
  adding to device set, 18
  addressable size, 20
  display size, 19, 20
  product/platform UIDs, 493
  Weather Client application and, 421, 436
Nokia E71, 22
  adding to device set, 17–18
  addressable size, 20
  Aino *v.*, 36
  Aino/Nokia E71 example, 35–47
  display size, 19, 20
  "Hello World" example, 23–26
  Image Viewer and, 283
  product/platform UIDs, 493
  TV Listings application and, 174–175
Nokia N95 8GB, 22
  adding to device set, 18
  addressable size, 20
  display size, 19, 20
  Media Console and, 238
  product/platform UIDs, 493
  Twitter client and, 324, 330, 336
Nokia N97
  application icons, 502
  contacts database and, 354
  on-screen touch keypad and, 421
  pre-installed applications for, 346
  product/platform UIDs, 493
notification names, 114, 127, 400
  in ApplicationFacade
    Image Viewer, 256–257
    Media Console, 194–195
    TV Listings application, 144
    Weather Client, 399–400
Notification UI component (listing 4–23), 104–105
notifications, 114–116
  declaring notification names in ApplicationFacade (listing 5–4), 114
  Observer and, 114–116
  sendNotification(), 115, 125, 328
Notifier class, 114, 115, 119
notifyObservers(), 115
Number, 527

O

Object, 527
object properties for System information (listing 3–18), 62
Objective C, 109
object-oriented programming (OOP), 33–74
  Aino/Nokia E71 example, 35–47
  encapsulation and, 35, 111
  inheritance and, 44, 112, 115, 220, 244, 246, 249
  polymorphism and, 35
  reasons for using, 35
Observer (design pattern), 110, 114–116
OEMs (original equipment manufacturers), 3, 7. *See also* Nokia; Sony Ericsson
On2 VP6, 10, 190
onClick(), 234
onComplete, 98, 203
onConnectionStatus(), 211–212
onContentLoad(), 102, 103
onDecrease, 101, 102
onDragOut(), 73, 511
onDragOver(), 73, 511, 525
onFocusChange, 90, 91, 92, 93
onFocusChangedHandler() (listing 4–13), 92–93
onFocusInHandler(), 80, 81, 86
onFocusOutHandler(), 80, 81, 86
onGetEntries(), 460, 461
onGetWeather(), 436
onID3, 186, 187, 188, 215, 216, 217, 529
onID3Complete(), 188, 215, 216, 217
onIncrease, 101, 102
onKeyDown(), 69–70, 520
  ExtendedKey and (listing 3–26), 71–72
  Key.getCode() (listing 3–24), 70
  listing 5–33, 135
onKeyUp(), 69–70, 520

www.it-ebooks.info

ExtendedKey and (listing 3–26), 71–72
Key.removeListener() (listing 3–25), 70–71
online library, 16, 17, 18, 105
onLoad, 186
onLoadComplete(), 152, 203, 204, 217, 268, 272, 394, 395, 410
onLoadFailed(), 410, 411
onMetaData, 182, 207, 209, 211, 212, 217, 526
onParseComplete(), 202, 203, 204, 404, 406, 410, 411, 416, 417
onPress(), 73, 473, 511, 525
onProgress, 98, 244
onRegister()
for MediaMediator, 230
for MediaProxy, 215
overriding, 132
for PlaylistMediator, 227
for SoundProxy, 216
onRelease(), 73, 168, 511, 525
onReleaseOutside(), 73, 74, 511, 525
onRollOut(), 73, 74, 511, 525
onRollOver(), 73, 74, 525
onScrollHandler(), 102, 103
onSelect, 81, 90, 91, 93, 436, 450
onSelectHandler(), 80, 86, 89, 395, 396, 397
handleNotification()and, 228
listing 4–14, 93–94
listing 7–58, 228
onSoftKeyDown(), 83, 450, 453, 463, 478, 479, 481, 482
onSoundComplete, 186, 187, 188, 529
onStatus, 180, 211, 214, 215, 255, 521, 526, 528, 535
onStreamStatus(), 211–212
onSubmit(), 435
onTextChange, 103, 104
OOP. *See* object-oriented programming
Open Screen Project, 6
open source. *See also* PureMVC; SWX
PureMVC and, 107
Qualcomm BREW and, 6
SWX and, 292
operating systems. *See also* Symbian OS
application signing and, 496
Flash Lite architecture and, 8
memory and, 29
OEMs and, 3
System.capabilities.os, 61, 62, 531
XP, Vista, Mac OS X, 14, 338
operators. *See* network operators
optional search parameters, 141
organize(), 353, 359, 360
original equipment manufacturers. *See* OEMs
os (System.capabilities.os), 61, 62, 531
overriding
getInstance() (listing 5–2), 112–113
initializeController() (listing 5–3), 113–114
initializeMacroCommand (listing 5–9), 118
onRegister() (listing 5–28), 132
switchOn/playRingtone methods (listings 3–10, 3–11), 46–47
Ovi, 506–507
Ovi Store, 506
owner (variable), 44

P

package (PKG) file, 491–495
Package Header (imageViewer.pkg), 492
packaging (Flash Lite content), 485–508
for different platforms, 486
Nokia Flash Packaging Tool, 486, 505–506
obstacles, 485–486
page number (search parameter), 141
parseCurrentConditions(), 405, 406, 407
parseForecastConditions(), 405, 407, 408
parseItemXMLNode(), 152, 153, 200, 201, 262
parseTimeline(), 304–305
parseWeather(), 406, 407, 408
parseXML(), 152, 204, 261, 262, 406
parsing JSON, 393–397. *See also specific parsers*
PAUSE, 193, 195, 225, 234
pause(), 180, 208, 209, 213, 223, 534
PauseCommand, 223
PCM (Pulse Code Modulation), 10
Persistency (Project Capuchin Service API), 440, 471–475
persistent data, 9, 12
Persistent Storage, 27
PGDN, 69, 520
PGUP, 69, 520
PKG file. *See* package file
platform (property), 41, 43, 44
PLAY, 193, 195, 225, 234, 235, 236
play(), 180, 182, 190, 191, 209, 475, 519, 524, 534
playback handling, 180
playback progress, 181–182
playback requests, 207–215
playback security, 176–177
PlayCommand (listings 7–48, 7–68), 222–223, 234
playFrom(), 213
PLAYLIST_ITEM_CHANGED, 235
PlayListItemVO (listing 7–11), 198–199
PlaylistLoader (listing 7–12), 199–200
PlaylistMediator (listings 7–54 to 7–59), 226–229
PlaylistParser (listing 7–13), 200–201
PlaylistProxy (Media Console), 197
creating (listings 7–6, 7–14, 7–15), 193–194, 201–203
loadPlaylist() (listing 7–16), 204
public methods, 199
PLAY_NEXT, 193, 195, 225, 234
PlayNextCommand, 223–224
PlayNow arena, 507–508

www.it-ebooks.info

PLAY_PREVIOUS, 193, 195, 225, 234
PlayPreviousCommand, 223–224
playRingtone(), 40, 46–47
playStream(), 213
polymorphism, 35
POST, 288
PrepareImagesCommand, 278–279
Pressed (UI component state), 78
previousResult(), 154
previousSection() (listing 5–27), 131–132, 133
private (keyword), 34
private keys, 498–499
processing power, 4
ProfileStyleVO, 302
progress indicators, 98–100
ProgressIndicator UI component (listing 4–18), 99–100
Project Capuchin initiative, 7
Project Capuchin Platform Service APIs, 439–484
  Bluetooth, 440, 448–454
  Calendar, 440, 454–457
  Contacts, 440, 457–461
  Core Project Capuchin MXP Service, 441
  I18n, 440, 465–469
  list of, 440
  Location, 440, 483
  Messaging, 440, 470–471
  methods, 441
  Multimedia, 440, 483
  online information, 439
  overview, 440
  Radio, 440, 475–483
Project panel
  Image Viewer, 259, 270, 277
  Media Console, 197, 226
  PureMVC example application, 127
  TV Listings application, 145
  Weather Client, 401, 423, 428
properties/methods/classes (ActionScript 2.0), 509–536. *See also specific properties*
proxies. *See also* SectionProxy
  Model and, 125–126
  registering, 125, 132–133
Proxy (design pattern), 111, 125–126
public (keyword), 34
public methods, IMobile interface, 40
Publish Settings panel, 25, 176, 177
publishing, to aggregators, 501
Pulse Code Modulation (PCM), 10
PureMVC, 107–137
  concept diagram, 110
  design patterns. *See* design patterns
  development frameworks and, 107–109
  key concepts, 111–136
  reliability, 109
  usability, 109
PureMVC example application, 126–136
  architecture, 126–127
  controller, 132–134
  in Device Central, 136
  model, 128–132
  view/mediators, 134–136
push buttons, 78–81
  states, 78
  using PushButton (listing 4–1), 79–81
Python, 109

Q

Qualcomm BREW, 6
Questions (dialogue component), 96, 97
QUIT, 144
QuitCommand, 116–117, 119, 165, 426
QWERTY keyboard, 36, 38, 40, 60, 62, 71
  hasQWERTYKeyboard, 62, 531
  hasQWERTYKeys, 40, 41, 42, 44, 45, 46

R

Radio (Project Capuchin Service API), 440, 475–483
radio buttons, 87–90, 95–96
Radio Data System (RDS), 481
Radio Times magazine, 140
RadioButton UI component (listing 4–5), 87–88
rate limiting, 291
RDA (Remote Device Access) service, 31
RDS (Radio Data System), 481
readFile(), 462
Real Time Messaging Protocol (RTMP), 10, 11, 190
Real Time Messaging Protocol Secure (RTMPS), 190, 191
Real Time Messaging Protocol Tunnel (RTMPT), 190, 191
Red5, 11
referrer, 140
referrer key, 140
registerCommand(), 119
registering
  commands, 119, 128
  mediators, 120–122
  proxies, 125, 132–133
  SectionProxy, 132–133
  StartupCommand and QuitCommand (listing 5–10), 119
registerMediator(), 120
RegisterNotification(), 370, 377
reliability (PureMVC), 109
Remote Device Access (RDA) service, 31
remove(), 95, 462
removeAllContacts(), 458
removeAllEvents(), 454
removeBroadcastMessageReceivedListener(), 470
removeClientDisconnectedListener(), 448
removeCommand(), 125
removeContact(), 458
removeCurrentConditions(), 430
removeDataReceivedListener(), 442, 445
removeEvent(), 454
removeEventListener(), 64, 65, 67
removeFileSystemChangedListener(), 462
removeForecast(), 432
removeMediator(), 125
removeMessageReceivedListener(), 448
removeMMSReceivedListener(), 470

www.it-ebooks.info

removeOrientationChanged Listener(), 442, 445
removeProxy(), 125
removeSMSReceived Listener(), 470
rename(), 462
rendering quality (Device Performance panel), 30
replies (Twitter), 288
RequestNotification(), 348
Required tab (Swf2Jar), 487–488
REST methods, 288–291
RESTART, 193, 195, 225, 234
RestartCommand, 224
resultDetails(), 152, 153, 154
resultHandler(), 308–309
results per page, 141
ResultsMediator (listing 6–26), 165, 170–172
ResultsProxy
  listing 6–4, 144, 145, 146–149
  listings 6–5 to 6–10, 151–155
retrieveMediator(), 125, 162, 282
retrieveMsg(), 155, 156
retrieveProxy(), 125, 319, 425
RetrieveUserDetailsCommand, 320
retrieving
  contact key values, 356
  contacts/databases/groups, 354–355
  information on apps, 343–344
  locations, 366
    automatic, 367–368
    canceling calls and, 368
  messages, 371–372
  meta data, 180, 378–383
  weather data, 403–404
returnTwoChars(), 419
ReturnValue, 341
retweeting, 287
reverse domain convention, 36
Review (Sony Ericsson submissions procedure), 507
RIGHT, 69, 520
ringtones, 7
_rowTimeout, 105
RTMP (Real Time Messaging Protocol), 10, 11, 190
RTMPS (Real Time Messaging Protocol Secure), 190, 191
RTMPT (Real Time Messaging Protocol Tunnel), 190, 191
Ruby, 109

S

S60 Platform
  packaging for, 486
  signing for, 500
  UIDs, 493
S60 Platform Services APIs, 337–384
  AppManager, 339, 343–347
  Calendar, 339, 347–353
  Contacts, 337, 353–360
  error codes, 342
  Landmarks, 339, 360–365
  library, 337, 338
  list of, 339
  Location, 339, 365–369
  Logging, 339, 384
  Media Management, 339, 378–383
  Messaging, 339, 369–378
  methods, 338–343
    asynchronous, 340–343
    synchronous, 339–340
  online information, 383, 384
  overview, 337–338
  Sensor Service, 339, 383–384
  SystemInfo, 339, 384
Sandbox type, 27, 176
SaveImageCommand, 280
saveMsg(), 155, 156
screen (property), 41, 44
screen size, 4
screenName(), 314
screensavers, 7
Scroll bar feature, 93
scrollable areas, 102–104
ScrollableArea UI component (listing 4–21), 102–103
_scrollTimeout, 104
SEARCH, 144
search parameters
  broadcaster, 140
  page number, 141
  referrer, 140
  results per page, 141
  search phrase, 141
  trim description, 141
search phrase, 141
Search Screen (wireframe), 397
SearchCommand (listing 6–20), 163–164
searchDevices(), 448
searchFrequency(), 476
SearchMediator, 433–436
SearchPanelMediator (listing 6–25), 165, 168–169
searchServer(), 448
SearchVal, 355
SectionProxy
  createData() (listing 5–26), 130–131, 132
  creating (listing 5–23), 129
  nextSection() (listing5–27), 131–132, 133
  overriding onRegister() (listing 5–28), 132
  previousSection() (listing 5–27), 131–132, 133
  private variables for (listing 5–24), 129–130
  registering, 125, 132–133
  setSection() (listing 5–25), 129, 130, 131
SectionVO, 124, 128–129
security (TV Listings application), 175–177
Security panel, 27, 176
Security tab (Swf2Jar), 489
seek(), 180
Selected, disabled, and highlighted, 89
Selected (UI component state), 89
Selected and disabled, 89
Selected and highlighted, 89
selectedIndex, 146, 152, 154, 205, 206, 264
SelectFirstImageCommand, 281
Selection, 528
SELECT_PLAYLIST_ITEM, 193, 195, 225, 228, 237
self-signing, 500
Send(), 370, 375

www.it-ebooks.info

sendEmail(), 40
sendFile(), 448, 450, 451, 453
sending messages, 375–376
sendMessage(), 448, 471
sendMMS(), 470
sendNotification(), 115, 125, 328
sendSMS(), 470
sendTxt(), 40
sendUpdate(), 157, 158
Sensor Service (S60 Platform Services API), 339, 383–384
Separator feature, 93
server response, for Kuala Lumpur, 387–388
set (keyword), 34
setBufferTime(), 181
setData(), 125, 148, 152, 268, 278, 417, 419
setFocusRectColor, 60
setFrequency(), 475
setHidden(), 462
SetInputTextType, 59–60
setInterval(), 65–66, 83, 99, 156, 181, 182, 214, 519
setLocale(), 466
setMute(), 476
setNextItem(), 199, 202, 203, 205, 223
setPan(), 189, 529
setPreset(), 476
setPresetName(), 476
setPreviousItem(), 199, 202, 203, 205, 223
setProgress(), 98, 99
setReadable(), 462
setResultAt(), 154
setSection() (listing 5–25), 129, 130, 131
setSelectedIndex(), 199, 202, 203, 205
setSoundMode(), 476
setTransform(), 189, 512, 529
setVolume(), 189, 218, 476, 529
setVolumeLevel(), 476
setWidth(), 99, 100, 104
setWritable(), 462
SharedObject, 254–255, 528
SharedObjectProxy, 266–268
SHIFT, 69, 520
showComponents, 233
showCurrentConditions(), 429
showForecast(), 432
ShowFriendsTimelineCommand, 321
SHOW_MAIN_MENU, 144, 161, 162, 167, 168, 169
ShowMainMenuCommand (listing 6–18), 162
SHOW_MEDIA, 232
SHOW_PLAYLIST, 232
showSearchPanel(), 434
SHOW_SEARCH_PANEL, 144, 161, 167, 168, 170, 171
ShowSearchPanelCommand (listing 6–17), 161–162
SHOW_USER, 309–310
ShowUserCommand, 321
ShowUserTimelineCommand, 322
ShowVO, 145, 149, 153, 159, 162, 163
shuffle option, 85
Signal class (listing 3–13), 51–52
signal levels, 52
signing applications. *See* application signing
signsis.exe, 499–500
SimpleCommand, 116–117
Simulate Performance (Device Performance panel), 30
simulation, 14, 26, 27, 29, 30, 99
single row lists (listings 4–6 to 4–14), 90–94
SingleImageCommand, 282
SingleImageMediator, 270–271
Singleton (design pattern), 110, 112, 115, 120
.sis file
  creating, 489
  makekeys.sis, 498–499
  makesis.exe tool, 489, 495, 496, 498
  process diagram, 490
  signsis.exe, 499–500
Sky Player, 139
Sky Songs, 179
Slider UI component (listing 4–20), 100–102
_sliderColor, 10
_sliderHighlightColor, 101
sliders, 100–102
SMAF (Synthetic [music] Mobile Application Format), 10, 63, 531
smartphones, 2, 4, 5, 337
SMS
  hasEmail and, 41, 43, 64, 531
  hasSMS and, 64, 531
  MessageType and, 371
  MessageTypeList and, 374
  messaging (Project Capuchin Service APIs), 440, 470–471
  send SMS from within an application (listing 10–39), 375–376
soft keys, 62, 71, 72, 81, 82–83, 97, 325, 335, 398, 464, 476, 481
softKeyCount, 62, 72, 531
SoftKeys UI component (listing 4–2), 82–83
Sony Ericsson, 7. *See also* Project Capuchin Platform Service APIs; UI components
  C510, 23
  http://developer.sonyericsson.com, 105
  PlayNow arena, 507–508
  submissions procedure, 507–508
  UI components, 76–105
Sorenson codec, 8, 10, 190
Sort object property, 342
sorting messages, 374–375
sortStrings(), 466
SortVO, 380
Sound class, 185, 186, 188, 529
  events, 529
  methods, 189, 529
  properties, 529
sound file (playing from library), 185
Sound object, 63
  audioMIMETypes, 63, 531
  Media Console and, 185–189
  Video object and, 188–189
soundContainer, 189, 192, 217, 229, 230, 231, 235, 236
SoundProxy (listings 7–36 to 7–43), 215–218
SPACE, 69, 520
speed (Device Performance panel), 30

www.it-ebooks.info

Spotify, 179
Stage class, 529–530
standalone versions (Flash Lite), 7
start(), for SoundProxy, 216–217
_startTimeout, 104
STARTUP, 114, 116, 144, 193
startup(), 115, 116, 193, 195, 220, 256, 257, 399
StartupCommand
  creating, 117–118
  Media Console (listing 7–44), 220
  TV Listings application (listing 6–14), 160
  Twitter client, 317
  Weather Client, 423
StartVibrate, 59
state-ready components, 77, 78
states (UI components), 78
static (keyword), 34
static heap value, 28–29, 54, 55
status, of messages, 377–378
status bar, 83–84
StatusBar UI component (listing 4–3), 83–84
statuses request, 291
StatusMediator (listing 7–72), 237–238
stop(), 185, 475, 476, 482, 519, 524, 534
stopTimer(), 157, 158
StopVibrate, 59
StoreLoginDetailsCommand, 320
streaming audio, 63
streaming FLV video, 63, 190–191
streaming MP3 audio, 191
String class, 530
submissions procedure (Sony Ericsson), 507–508
super(), 44, 45, 122, 244, 245, 246, 250
svg2svgt tool, 502
SWF Audio, 8
SWF format, 5
Swf2Jar, 486–489
switchOff(), 40
switchOn(), 40, 46–47
SWX Data Analyzer, 298
SWX format, 292–298
SWX gateway, 296
SWX PHP, 292–294
SWX RPC, 29, 293, 298
SWX Service Explorer, 297–298
SWX TwitterProxy, 306–307
Symbian OS, 337
  releases, 337–338
  UIDs and, 489
Symbian S60 Platform. *See* S60 Platform
Symbian Signed program, 489, 500, 501
synchronous methods, 339–340
Synthetic [music] Mobile Application Format (SMAF), 10, 63, 531
System.capabilities, 10, 60–64, 192, 531
  audioMIMETypes, 63, 531
  communication features, 64
  external data support, 62
  has4WayKeyAS, 62, 531
  hasAudio, 63, 192, 531
  hasCMIDI, 63, 531
  hasCompoundSound, 63, 531
  hasDataLoading, 62, 531
  hasEmail, 41, 43, 64, 531
  hasEmbeddedVideo, 63, 192, 531
  hasMappableSoftKeys, 62, 531
  hasMFI, 63, 531
  hasMIDI, 63, 531
  hasMMS, 64
  hasMouse, 63
  hasMP3, 63, 192, 531
  hasQWERTYKeyboard, 62, 531
  hasSharedObjects, 62, 531
  hasSMAF, 63, 531
  hasSMS, 64, 531
  hasStreamingAudio, 63, 192, 531
  hasStreamingVideo, 63, 192, 531
  hasStylus, 63, 72, 531
  hasXMLSocket, 62, 531
  imageMIMETypes, 63, 242, 531
  information about system, 61–62
  key input/device interaction, 62–63
  language, 59, 61, 62
  media capabilities, 63–64
  MIMETypes, 64, 531
  object properties for System information (listing 3–18), 62
  os, 61, 62, 531
  softKeyCount, 62, 72, 531
  version, 61, 62, 531
  videoMIMETypes, 64, 192, 531
SystemInfo (S60 Platform Services API), 339, 384
System.Security, 532

T

TAB, 69, 520
Target Platform (imageViewer.pkg), 493
testing content. *See also* emulator
  Automated Testing section, 26
  on mobile devices, 30–31
text
  dynamic, 77
  input, 77
  static, 77
  support, 11
textColor, 81, 83, 105, 533
TextField component, 24, 59, 76, 77, 532–533
  events, 533
  methods, 533
  properties, 532–533
  TextHelper class and, 326, 327
  UserLoginMediator and, 324, 325
  UserStatusMediator and, 331, 332, 334
TextFormat, 534
32KB, 29
3G data network, 2
Thumbplay, 501
tickComplete()
  dispatchEvent and (listing 3–21), 66–67
  init and, 66
  listing 3–20, 66
tight-coupling, 109
TimelineParser, 304–305
timelines (Twitter), 287
TimelineVO, 303
timeoutHandler(), 313
TimeProxy, 401, 417–420

www.it-ebooks.info

Timer class, 65–67
  clearInterval(), 65–66
  creating (listing 3–19), 65
  setInterval(), 65–66
  tickComplete()
    dispatchEvent and (listing 3–21), 66–67
    init and, 66
    listing 3–20, 66
  for TV Listings application (listing 6–12), 156–157
TimerListener class (listing 3–22), 67
Title component, 84, 86
To-Do note, 349, 350
toggleSound(), 218, 219, 224
TOGGLE_SOUND, 193, 195, 225, 234
ToggleSoundCommand, 193, 196, 224
tools (for Flash Lite mobile application development), 13–31. *See also* Device Central; Flash CS4
touch interaction, 72–74
  buttons and, 72–74
  hasTouchUI and, 40, 41, 42, 44, 45, 46
Trace(), 155, 366, 367, 368, 395
_trackColor, 101
trends (Twitter), 287
trial version (Flash CS4), 15
trim description, 141
TV Genius search engine API, 139, 140–141
  calls
    parameters, 140–141
    response format for, 141
TV Listings application, 139–177
  ApplicationFacade, 143–144
  concept for, 139–140
  conceptual diagrams
    controller, 159
    model, 145
    view, 166
  controller, 158–165
  in Device Central, 174–175
  .fla file, 143
  functional requirements, 142
  model, 144–158
  Nokia E71 and, 174–175
  security issues, 175–177
  view/mediators, 165–175
  XML data and, 149–158
Tweets, 287
Twitter, 286–288
  API methods, 288–291
  status, 287
  terminology, 286–288
  user account, 286
Twitter client, 285–336
  ApplicationFacade, 299–300
  authentication and, 318–319
  conceptual diagrams
    controller, 316
    model, 301
    view, 323
  controller, 316–322
  in Device Central, 336
  .fla file, 299
  JSON and, 288, 295
  model, 300–315
  Nokia N95 8GB and, 324, 330, 336
  view/mediators, 322–336
TwitterProxy (listings 9–8 to 9–25), 303–315
  creating, 305–315
  importing classes into, 306
  initCall() for, 307–308
  methods, 305
  parsers, 303–304, 309
  resultHandler(), 308–309
  SWX, 306–307
  SWX call arguments for, 307

U

UI (user interface) components, 75–105. *See also specific components*
  Forum Nokia, 76–78, 105
  Sony Ericsson, 76–105
  state-ready, 77, 78
  states, 78
UIDs (unique identifiers)
  Nokia devices, 493
  S60 Platform, 493
  Symbian Signed Web site and, 489
"Understanding the Mobile Ecosystem," 4
unique identifiers. *See* UIDs
unloadImages(), 98, 104
untrusted content, 497–498
UP (static property), 69, 520
UPDATE, 311–312
updateContact(), 458
updateEvent(), 454
updateSoftkeys(), 228, 229, 270, 272, 299, 325, 328, 329, 331, 334, 335, 400, 429, 430, 432, 434, 435
UpdateStatusCommand, 322
updateTimeStr(), 419, 420
usability (PureMVC), 109
usePreset(), 476
user credentials, verifiying, 289–290
user interface components. *See* UI components
user notification, 104–105
UserLoginMediator (listings 9–38 to 9–42), 324–330
UserParser, 303–304
UserStatusMediator (listings 9–43 to 9–47), 330–334
USERS_TIMELINE, 310–311
userTimeline, 294–296
UserVO, 302
UTF-8, 11, 363

V

vCalendar, 348–349
vCard files, 358–359
Vector Font Data, 8
Vendor Name (imageViewer.pkg), 494
VERIFY_CREDENTIALS, 312
VerifyCredentialsCommand, 319–320
verifying user credentials, 289–290
version (System.capabilities), 61, 62, 531
Video class, 184, 534–535
Video object
  creation, 184–185

www.it-ebooks.info

Sound object and, 188–189
Video section (Device Profile panel), 17
video support, 10–11
videoMIMETypes, 64, 192, 531
VIDEO_STREAM_CREATED, 235
View. *See also* PureMVC
Image Viewer, 268–275
Media Console, 224–238
mediators and, 119–124
in MVC, 109
PureMVC example application, 134–136
TV Listings application, 165–175
Twitter client, 322–336
Weather Client application, 426–436
ViewCurrentConditionsCommand, 425–426
VIEW_DETAILS, 144, 162, 166, 170, 171, 172
ViewDetailsCommand (listing 6–19), 162–163
ViewForecastConditionsCommand, 426
ViewImageGridCommand, 281
ViewPrepCommand, 117, 118, 120, 122, 126, 127, 132, 219, 220, 226, 276, 277, 316, 317, 318, 422, 423, 424
Image Viewer (listing 8–54), 277–278
Media Console (listing 7–46), 221–222
TV Listings application (listing 6–16), 161
Twitter client, 318
Weather Client, 424
ViewSingleImageCommand, 282
Vista (Windows Vista), 14, 338
visual indicators, 98–100
VP6 (On2), 10, 190

W

wait indicators, 100
WaitIndicator UI component (listing 4–19), 100
WaitMediator, 274–275
Warnings (dialogue component), 96, 97
Weather Client application, 385–437
ApplicationFacade, 399–400
conceptual diagrams
controller, 422
model, 401
view, 427
controller, 421–426
in Device Central, 436
.fla file, 399
Google Map API, 390–397
Google Weather API, 386–390
Location Services and, 365
model, 400–421
Nokia 5800 XpressMusic and, 421, 436
retrieving weather data, 403–404
view/mediators, 426–436
wireframes and, 397–398
weather icons, 389–390
WeatherVO, 402–403
Web section (Device Profile panel), 17
Weiss, Tom, 139
Wi-Fi networks, 2
wild card, 112
Windows Mobile platform
packaging for, 486
signing for, 500
Windows Vista, 14, 338
Windows XP, 14, 338
wireframes, 397–398
_wrapTimeout, 105
writeFile(), 462

X

XML API reply, 387–388
XML class, 535
XML data (TV Listings application), 149–158
XML objects, 149–150
XML support, 12
XML/KML output (Google Map API), 391–392
xml.load(), 147, 150, 152, 176, 200, 260, 261, 403
XMLNode, 150–153, 177, 536
XMLSocket, 536
XP (Windows XP), 14, 338
XPathAPI, 150–151
XPathAPI.selectNodeList(), 150, 151, 152, 200, 262, 407
XPathAPI.selectSingleNode(), 150, 151, 153, 201, 263, 406, 407, 408

Y

YouTube, 139, 285

Z

Zed, 501

Download from Wow! eBook <www.wowebook.com>

www.it-ebooks.info

Discover how to create Flash Lite mobile apps from the ground up

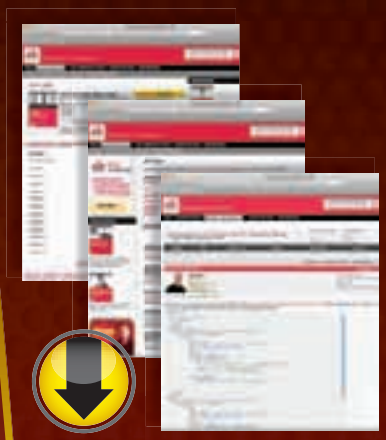
Adobe Flash is an ideal choice for developing rich interactive content for "Flash-enabled" mobile devices; and with this book, you'll learn how to create unique applications with Flash Lite. Through a series of code samples and extensive example applications, you'll explore the core concepts, key features, and best practices of the Flash Lite player. Coverage reveals various ways to develop Flash mobile content, create applications with a cross-platform programming framework based on the Model, View and Controller concept, and use a number of open web and device manufacturer service APIs.

Professional Flash Lite Mobile Development:

- Looks at Adobe Flash Professional and Adobe Device Central, the essential tools used in developing and testing Flash mobile device content
- Explores object-oriented programming as well as the PureMVC framework
- Shares techniques for extending the capabilities of mobile devices
- Provides detailed insight into using audio, video, and images in applications
- Introduces ways to extend Flash mobile apps beyond the features of the Flash Lite Player
- Walks you through packaging and distributing complete Flash Lite mobile apps to different mobile platforms

Jermaine G. Anderson is a Flash Platform Developer who has developed a wide range of concept mobile applications using Flash Lite. He works for a leading entertainment and communications company, pushing the boundaries of the Adobe Flash Platform on multiple devices.

Wrox Professional guides are planned and written by working programmers to meet the real-world needs of programmers, developers, and IT professionals. Focused and relevant, they address the issues technology professionals face every day. They provide examples, practical solutions, and expert education in new technologies, all designed to help programmers do a better job.



wrox.com

Programmer Forums

Join our Programmer to Programmer forums to ask and answer programming questions about this book, join discussions on the hottest topics in the industry, and connect with fellow programmers from around the world.

Code Downloads

Take advantage of free code samples from this book, as well as code samples from hundreds of other books, all ready to use.

Read More

Find articles, ebooks, sample chapters, and tables of contents for hundreds of books, and more reference resources on programming topics that matter to you.

ISBN 978-0-470-54748-9
54499
9 780470 547489

Wrox™
An Imprint of
WILEY

Internet / Mobile Development

$44.99 USA
$53.99 CAN

www.it-ebooks.info