Carmen Delessio

24 Proven One-hour Lessons

Sams Teach Yourself

Google™ TV App Development in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for *Sams Teach Yourself Google TV™ App Development in 24 Hours*

"Although the TV has been a networked device for many years only modern platforms, like Google TV, offer developers the ability to create powerful apps. This book is a great practical guide on how to develop your connected TV app."

—Alberto Escarlate, co-founder, Sojo Studios

"Mr. Delessio may have written this book for developers, but it's got great value for interaction designers and product managers needing to understand product creation for interactive TV and multi-screen platforms. If you're responsible for strategizing, requirements or building Google TV, second screen, or device-adaptive applications, you need this knowledge. Though the many code examples are clearly for developers, technical product managers and designers with basic HTML/CSS/Java knowledge can follow along. Interactive television has been a dream for decades. It's actually here now. And this is how to get on board."

—Scott Germaise, Internet entrepreneur and co-founder, About.com, KeepHoldings.com

"Up until this point, I only tested apps for the Google TV; after reading this book, I now have the tools and confidence to start developing my own application."

—Chris Hollis, Google TV Friends

"This book gently steps an Android developer back from the intimate smartphone experience to the big-screen view from the living room couch, and packs plenty of solid, useful information into 24 hours."

—Jonathan Taylor, VP, Mobile Technology, priceline.com

This page intentionally left blank

Carmen Delessio

Sams **Teach Yourself**

Google TV™ App Development

in **24 Hours**

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Google TV™ App Development in 24 Hours

Copyright © 2013 by Pearson Education

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33603-4

ISBN-10: 0-672-33603-4

Library of Congress Cataloging-in-Publication data is on file.

First Printing January 2013

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Several images in this book use scenes from the online movie Big Buck Bunny to illustrate the use of online video in Google TV apps. This movie and related material is distributed under a Creative Commons license. For more information on the movie go to http://www.bigbuckbunny.org/.

Blender Foundation | www.blender.org

© copyright 2008, Blender Foundation / www.bigbuckbunny.org

Some images in this book are reproduced or are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License. See https://developers.google.com/readme/policies.

Screenshots of Google products follow these guidelines:

http://www.google.com/permissions/using-product-graphics.html

The following are registered trademarks of Google:

Android™, Google Play™, YouTube Leanback™ service, YouTube Remote™ mobile app, Google TV Remote™ mobile app, Google TV™ platform, Google Chrome™ browser.

Google and the Google logo are registered trademarks of Google Inc., used with permission.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Pearson offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

international@pearsoned.com

Acquisitions Editor
Laura Lewin

Development Editor
Michael Thurston

Managing Editor
Sandra Schroeder

Project Editor
Seth Kerney

Copy Editor
Barbara Hacha

Indexer
Brad Herriman

Proofreader
Jess DeGabriele

Technical Editors
Romin Irani
Ronan Schwarz
Max Tomlinson

Publishing Coordinator
Olivia Basegio

Book Designer
Gary Adair

Composition
Bronkella Publishing

Contents at a Glance

HOUR 1 Developing for Google TV and the 10-Foot User Experience ... 1
2 Optimizing Web Pages for Google TV ... 15
3 Using Google's Video Templates ... 31
4 HTML5 and jQuery UI Library for Google TV ... 45
5 Using the Google TV Closure UI Library ... 61
6 Creating a Video Sitemap ... 83
7 Android and Google TV ... 95
8 Using Android Layouts for Your Google TV App ... 113
9 Optimizing UI Components for Google TV ... 129
10 Organizing Google TV Apps Using the ActionBar ... 149
11 Understanding Activities and Fragments in a Complex App ... 163
12 Using Specialized Fragments ... 181
13 Handling D-Pad and Key Events ... 195
14 Accessing Remote Content for a Google TV App ... 207
15 Storing Data with SQLite ... 225
16 Creating a Content Provider ... 239
17 Using Cursors and CursorLoaders ... 257
18 Developing a Complete App ... 269
19 Mastering the Android Manifest File ... 289
20 Using the Channel Listing Provider ... 301
21 Using Second Screen Apps with Google TV ... 315
22 Examining an Example Second Screen App ... 325
23 Developing Second Screen Apps ... 337
24 Working with Anymote and Pairing Protocols ... 353
Index ... 367

Table of Contents

Hour 1: Developing for Google TV and the 10-Foot User Experience **1**

Google TV UI ... 1
TV as a Device ... 3
Google TV Apps ... 6
10-Foot User Experience ... 10
Summary ... 13
Q&A ... 13
Workshop ... 13
Exercises ... 14

Hour 2: Optimizing Web Pages for Google TV **15**

Displaying Content on TV Screens ... 15
Playing Video ... 20
Displaying Text and Color ... 22
Navigating the Big Screen ... 25
Summary ... 29
Q&A ... 29
Workshop ... 29
Exercises ... 30

Hour 3: Using Google's Video Templates **31**

Google Video Templates ... 31
Examining the Template Structure ... 33
Understanding the JavaScript for Template 2 ... 36
Using Templates to Show Your Own Videos ... 38
Retrieving Videos Using AJAX ... 41
Summary ... 42
Q&A ... 42
Workshop ... 42
Exercises ... 43

Hour 4: HTML5 and jQuery UI Library for Google TV **45**

Video Properties and Controls ... 45
Adding Video Controls ... 46
Adding Some Style ... 50
Using the jQuery GTV UI Library ... 52
Summary ... 59
Q&A ... 59
Workshop ... 59
Exercises ... 60

Hour 5: Using the Google TV Closure UI Library **61**

Using Closure for Google TV Web Apps ... 61
Using the Library to Create an HTML Page ... 63
Buttons, Lightboxes, and Other Components ... 66
Summary ... 80
Q&A ... 80
Workshop ... 80
Exercises ... 81

Hour 6: Creating a Video Sitemap **83**

Searching on Google TV ... 83
Sitemaps and Google Webmaster Tools ... 84
Adding Details to Video Sitemaps ... 85
Tags That Provide Additional Info on Google TV ... 87
Summary ... 92
Q&A ... 92
Workshop ... 92
Exercises ... 93

Hour 7: Android and Google TV **95**

Android, the Google TV Operating System ... 95
Understanding the Development Environment ... 97
Testing Your Apps ... 100
Setting Up an Android Project in Eclipse ... 104

Summary 112
Q&A 112
Workshop 112
Exercises 112

Hour 8: Using Android Layouts for Your Google TV App **113**
Layout Basics with LinearLayout 113
Laying Out Widgets 119
FrameLayout 122
RelativeLayout 124
Summary 127
Q&A 127
Workshop 127
Exercises 128

Hour 9: Optimizing UI Components for Google TV **129**
Form Widgets 129
Using ImageViews 137
Using VideoViews 140
Putting Button Overlays on a Video 142
Summary 146
Q&A 146
Workshop 146
Exercises 147

Hour 10: Organizing Google TV Apps Using the ActionBar **149**
Implementing an ActionBar 149
Adding Tabs 151
Adding ActionItems and MenuItems 153
Using the LeftNavBar Library 155
Putting It All Together 156
Summary 161
Q&A 161
Workshop 161
Exercises 162

Hour 11: Understanding Activities and Fragments in a Complex App 163

Understanding the Activity Life Cycle 163
Understanding Fragments 165
Swapping Fragments in an Activity 166
Using Fragments with ActionBar 174
Summary 178
Q&A 178
Workshop 178
Exercises 179

Hour 12: Using Specialized Fragments 181

DialogFragments 181
ListFragments 186
Grids and Galleries 188
Summary 193
Q&A 193
Workshop 193
Exercises 194

Hour 13: Handling D-Pad and Key Events 195

Designing for the D-Pad 195
Controlling Focus 197
Programming D-Pad Key Events 199
Summary 205
Q&A 205
Workshop 205
Exercises 206

Hour 14: Accessing Remote Content for a Google TV App 207

Displaying Remote Images and Videos 207
Fetching Remote Data 212
Using and Parsing JSON Formatted Data 215
Putting the Pieces Together in the App 218
Summary 222
Q&A 222
Workshop 222
Exercises 223

Hour 15: Storing Data with SQLite **225**

Organizing a Database with Tables ... 225
Managing Data with SQLiteOpenHelper ... 226
Adding, Deleting, and Updating Data ... 228
Querying Data and Using Cursors ... 231
Using a Database in the App ... 233
Summary ... 237
Q&A ... 237
Workshop ... 237
Exercises ... 238

Hour 16: Creating a Content Provider **239**

Using a URI for Data Retrieval ... 239
Building a ContentProvider ... 240
Using PhotoProvider in Our App ... 248
Requesting a File from a ContentProvider ... 249
Summary ... 255
Q&A ... 255
Workshop ... 255
Exercises ... 256

Hour 17: Using Loaders and CursorLoaders **257**

How Loaders Work ... 257
Loader Classes ... 258
Loader States and LoaderManager.Callbacks ... 258
Adding Loaders to ImageViewFragment ... 260
Summary ... 266
Q&A ... 266
Workshop ... 266
Exercises ... 267

Hour 18: Developing a Complete App **269**

Determining What the App Will Do ... 269
Developing the GridFragment ... 274
Enhancing the Activity Class ... 281
Adding the ActionBar ... 282

Summary ... 286
Q&A ... 286
Workshop ... 286
Exercises ... 287

Hour 19: Mastering the Android Manifest File **289**
Android Manifest Basics ... 289
Understanding Intents ... 292
Adding Google TV Specifics to the Manifest File ... 294
Publishing Your App ... 296
Summary ... 298
Q&A ... 298
Workshop ... 298
Exercises ... 299

Hour 20: Using the Channel Listing Provider **301**
Getting Data from the Channel Listing Provider ... 301
Listing All Channels ... 305
Changing the Channel ... 307
Understanding Audio Focus ... 310
Summary ... 313
Q&A ... 313
Workshop ... 313
Exercises ... 314

Hour 21: Using Second Screen Apps with Google TV **315**
How Second Screen Apps Are Used ... 315
Types of Second Screen Apps ... 316
User Interaction with Second Screen Apps ... 316
Connecting a TV with a Second Screen ... 318
Summary ... 323
Q&A ... 323
Workshop ... 323
Exercises ... 324

Hour 22: Examining an Example Second Screen App 325
Preparing the Blackjack Second Screen App 325
Running the Blackjack App 328
Exploring the Blackjack App Code 330
Exploring the Anymote Library Code 332
Summary 334
Q&A 334
Workshop 334
Exercises 335

Hour 23: Developing Second Screen Apps 337
Flinging a URL 337
Sending KeyEvents to the TV 343
Sending Mouse Events to the TV 346
Summary 350
Q&A 350
Workshop 350
Exercises 351

Hour 24: Working with Anymote and Pairing Protocols 353
The Anymote Protocol 353
Pairing Protocol 358
Using Anymote and Pairing in a Chrome Extension 360
Summary 364
Q&A 364
Workshop 364
Exercises 365

Index 367

Preface

A book about Google TV—really?

Yes, really. It seems like we've been waiting for Smart TVs or Interactive TVs for quite a long time. Streaming video and movies on demand has been a concept for many years that has now become reality. Smart TVs with web browsing and apps is the next step and Google TV is there.

With the release of the Android HoneyComb version for Google TV and, more importantly, the creation of an app market in late 2011, Google has created a great ecosystem for television development. The earliest version of Google TV did not provide a way for developers to create and publish apps. That changed with the addition of the Google Play market.

New devices such as the Vizio CoStar and TVs from Sony and Samsung are rolling out with Google TV onboard. Rather than develop their own proprietary systems, TV manufacturers can rely on Google and the Android operating system to provide smart TVs to consumers.

Why Google TV Is Important

Google has used the Android operating system as a common platform across phone manufacturers. Google TV plays the same role for TV manufacturers. There is no common cross-manufacturer platform for TV development. Google TV fills that void.

Google TV provides a new app market.

The iPhone showed that an app marketplace was a key success factor for a new device. The iPad showed that a new form factor with a different size and shape makes a difference for both development of apps and how consumers use them.

Google TV includes both a new market for apps and a new experience for enjoying apps. This is a new opportunity for developers. It is a great time to jump into smart TV development on a Google TV.

Additionally, there really is something new with Google TV: Second-screen apps, in which a phone or a tablet interacts with a TV, providing exciting new opportunities for developers and app designers. Second-screen apps are covered in detail in this book from a conceptual and development perspective. This book also covers the Anymote Library and how to use it with second-screen apps. Your phone and tablet apps will discover, connect to, and control Google TVs.

Who This Book Is For

If you are a web developer who knows HTML, CSS, and JavaScript, this book can help you learn how to optimize your websites and apps for Google TV. There is a jQuery Google TV Library and a Closure JavaScript-based library that can be used with Google TV; both are covered in this book.

If you are a Java programmer with an interest in TV development, this book will cover the basics of Android development so that you can create sophisticated TV apps on Android.

If you are an Android developer and want to learn the fine points of TV development, this book will take you through the details. If you want to take advantage of your Android skills to create second-screen apps, you will be able to rely on this book for both the concepts and implementation details you need.

How This Book Is Organized

This book covers developing web apps, Android apps, and second-screen apps for Google TV.

Hour 1 introduces basic concepts of designing apps for what has come to be called the 10-foot user experience. When building a Google TV app, developers must understand the idea that the user sits 10 feet away from a TV screen.

Hours 2 through 6 cover Google TV development from a web app perspective. Optimizing a site and using Google TV specific libraries are covered.

Hours 7 through 19 are about developing Android apps on Google TV. Android app development building blocks are covered with an emphasis on developing for Google TV. Over the course of Hours 15 through 18, a sophisticated app is developed that displays images from Facebook pages on a TV.

Hour 20 shows how to get information on available TV channels and how to change channels from an app using the Channel Listing provider unique to Google TV.

Hours 21 through 24 focus on second-screen apps. Hour 21 introduces second-screen apps and discusses how they work and how to handle potential design challenges. Hour 22 shows how to download, install, and run a sample app and shows how that app is constructed. In Hour 23, a new second screen app is developed. In Hour 24, the underlying concepts and protocols for communicating with a TV are covered. That knowledge can be used as the basis for other devices or apps to communicate with the TV.

About the Author

Carmen Delessio is an experienced application developer who has worked as a developer, technical architect, and CTO in large and small organizations.

Carmen developed the award-winning “BFF Photo” Android app. The app has more than 300,000 downloads and was the winner of the Sprint App Challenge contest in the Social Networking category.

Carmen began his online development career at Prodigy, where he worked on early Internet applications, shopping apps, and fantasy baseball.

He has written for Mashable, AndroidGuys, and Screenitup.com. Screenitup focuses on second-screen TV apps.

He is a graduate of Manhattanville College and lives in Pound Ridge, New York, with his wife, Amy, and daughter, Natalie.

Dedication

For Amy and Natalie.

Acknowledgments

This book would not exist without the help and guidance of the team at Pearson (Sams Publishing). Thanks to Laura Lewin for constant encouragement, Olivia Basegio for her continued work on the project, and Trina McDonald for staying in contact and suggesting the book. Particular thanks go to Michael Thurston for the insightful comments and suggestions during editing. The thoroughness and clarity that were added make this a much better book.

Thanks, Amy and Natalie, for enjoying Saturday morning ballet classes, birthday parties, and other activities while I was writing.

We Want to Hear from You

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write directly to let us know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book's title and author, as well as your name and contact information.

Email: feedback@samspublishing.com

Mail: Reader Feedback
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at **www.informit.com/register** for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

HOUR 23
Developing Second Screen Apps

What You'll Learn in This Hour:

- ▶ How to fling a URL
- ▶ How to send KeyEvents to the TV
- ▶ How to send mouse events to the TV

In Hour 22, "Examining an Example Second Screen App," you installed the example Blackjack second screen app. In doing so, you downloaded and installed the Anymote Library and set up three Eclipse projects. Those projects were the Anymote Library, the remote control app, and the TV app.

In this hour, we will develop several second screen apps. The first app takes a URL entered on the remote device and opens the corresponding web page in the Chrome browser on Google TV. That represents a simple content app that pairs with the TV, but runs only on the remote. It provides an example of a basic remote app and uses the ability to fling an Android Intent to the TV. Apps for keyboard input and mouse events will also be developed. We'll use the phone's Accelerometer as the input device for sending mouse events to the TV. These example apps provide a foundation for creating more sophisticated second screen apps.

Flinging a URL

By installing and running the Blackjack second screen app in Hour 22, you added the Anymote Library to your Eclipse projects. We'll use the Anymote Library and develop a remote app that takes a URL as input on an Android phone and opens the website on the Chrome browser on a Google TV. Because we can use an Android Intent to show the website on the TV, there is no need to create a separate TV app.

An App for Displaying Websites on a Google TV

We'll start with an Android app that includes an `EditText` field for input and a button to initiate an action. We'll make it a Google TV remote app by doing the following:

- Adding the Anymote Library
- Implementing the Activity as an `AnymoteClientService ClientListener`
- Binding to the Anymote Service
- Modifying the Android manifest to support the Anymote Library

All these things were done in the code for the Blackjack remote app, and we will use that as our model.

The input field for this app will accept a URL. The button in the activity reads the URL and attempts to send it to the TV. Figure 23.1 shows the remote user interface.

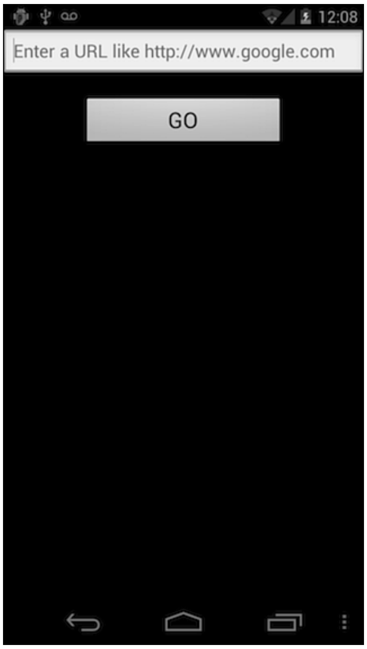

FIGURE 23.1
Send a URL to the Chrome Browser remotely.

The Anymote Library is added to the working Eclipse project, as it was in Hour 22.

For this app, we need only the remote app, and it will consist of a single Activity. Each remote app in this hour has a similar code structure:

- An Activity that implements the `AnymoteClientService ClientListener`
- A declaration of a `ServiceConnection` to bind to the Anymote Service
- The Activity's `onCreate` method that includes the binding to the Anymote Service
- The implementation of the three methods required for `ClientListener`

Our Activity is called `ExampleOneActivity`. Similar to the `BlackJackRemoteActivity`, it is declared as follows:

```
public class ExampleOneActivity extends Activity implements ClientListener
```

Listing 23.1 shows the declaration of the Anymote sender and the service connection. A `ServiceConnection` is an `Interface` for monitoring the state of an application service. In our case, the service is the `AnymoteClientService`.

LISTING 23.1 ServiceConnection Declaration

```
private AnymoteSender anymoteSender;
private ServiceConnection mConnection = new ServiceConnection() {
  public void onServiceConnected(ComponentName name, IBinder service) {
     mAnymoteClientService=((AnymoteClientService.AnymoteClientServiceBinder)
     service).getService();
     mAnymoteClientService.attachClientListener(ExampleOneActivity.this);
  }
  public void onServiceDisconnected(ComponentName name) {
     mAnymoteClientService.detachClientListener(ExampleOneActivity.this);
      mAnymoteClientService = null;
  }
};
```

In the `onCreate` method, we bind the `AnymoteClientService` to the Activity. The code to do that is shown next. An `Intent` is created using the activity and the `AnymoteClientService` class. A call to `bindService` binds the intent to the `ServiceConnecton mService` that we declared earlier. The flag `BIND_AUTO_CREATE` indicates that the service should be created.

```
Intent intent = new Intent(ExampleOneActivity.this,
                    AnymoteClientService.class);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

Listing 23.2 shows the complete `onCreate` method for the Activity.

LISTING 23.2 Remote Activity onCreate Method

```
1:  @Override
2:  public void onCreate(Bundle savedInstanceState) {
3:    super.onCreate(savedInstanceState);
4:    mContext = this;
5:    setContentView(R.layout.main);
6:    progressBar = (ProgressBar) findViewById(R.id.a_progressbar);
7:    progressBar.setVisibility(View.VISIBLE);
8:    Button go = (Button) findViewById(R.id.go);
9:    final EditText destination = (EditText) findViewById(R.id.destination);
10:   go.setOnClickListener(new OnClickListener() {
11:     @Override
12:     public void onClick(View v) {
13:       String url = destination.getText().toString();
14:       final Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
15:       anymoteSender.sendUrl (intent.toUri(Intent.URI_INTENT_SCHEME));
16:     }
17:   });
18:   handler = new Handler();
19:   Intent intent = new Intent(ExampleOneActivity.this, AnymoteClientService.
class);
20:   bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
21: }
```

Lines 19 and 20 in Listing 23.2 show the binding of the `AnymoteClientService` to the Activity.

Lines 5 to 9 define the UI components. A `ProgressBar` is shown while pairing occurs. An `EditText` and `Button` are defined. A URL can be entered into the `EditText`. The `Go` Button initiates the action of displaying the URL on the TV. Figure 23.1 showed what these look like on the remote.

Lines 10 to 17 define the action that occurs when the Button is pressed. The value in the `EditText` field is read in line 13. In line 14, that value is used to create an `Intent`. Line 15 sends the `Intent` to the Google TV using the `anymoteSender`. The TV will launch the Chrome browser and display the page associated with the URL.

The work of the app is done in the `onCreate` method, but the setup of the AnymoteService and library is required to make it happen. Finally, the three methods required for the ClientListener must be implemented. Those are the `onConnected`, `onDisconnected`, and `onConnectionFailed` methods. They are shown in Listing 23.3. Because the Activity implements ClientListener, we can say that it is a ClientListener.

We are relying on the Anymote Library for the user interface for pairing. Our app controls the `ProgressBar` and provides the UI for navigating to a URL. Listing 23.3 shows that we hide the `ProgressBar` when the connection is made. The field `anymoteSender` is populated in the `onConnected` method on line 3.

LISTING 23.3 Required Methods for ClientListener

```
1:   @Override
2:   public void onConnected(final AnymoteSender anymoteSender) {
3:     this.anymoteSender = anymoteSender;
4:     handler.post(new Runnable() {
5:       public void run() {
6:         progressBar.setVisibility(View.INVISIBLE);
7:       }
8:     });
9:   }
10:  @Override
11:  public void onDisconnected() {
12:    this.anymoteSender = null;
13:  }
14:  @Override
15:  public void onConnectionFailed() {
16:    System.out.println(“connection failed”);
17:  }
```

The `onDestroy` method for the Activity is used to clean up any outstanding resources when the Activity is destroyed. In Listing 23.4, we use the `onDestroy` method to detach and unbind the Anymote services from the current Activity.

LISTING 23.4 Cleaning Up in the onDestroy Method

```
@Override
protected void onDestroy() {
  if (mAnymoteClientService != null) {
    mAnymoteClientService.detachClientListener(this);
  }
  unbindService(mConnection);
  super.onDestroy();
}
```

The Anymote Library handles the UI for discovery and pairing. For this remote app, no companion TV app is required. The native functionality of flinging an Intent to the TV provides the functionality required.

An App for Showing a Facebook Image on a Google TV

We will create another second content app that flings a URL to the TV that is based on the work we did in Hour 18, “Developing a Complete App.” In that hour, we created a Google TV app that showed random images from a Facebook page. We will turn that app into a remote content app using the Anymote protocol. The phone will show a random image when a button is pressed. The TV will show the same image.

The `onCreate` method for the app shows how we make this happen. It is shown in Listing 23.5. The basic infrastructure of a remote app is followed. The Activity extends `ClientListener` and binds to the `AnymoteClientService`.

In the app in Hour 17, “Using Cursors and CursorLoaders,” we retrieved a list of photos from Facebook. We parsed the data and selected a random image to display. We select the random URL as a `String` in line 16 of Listing 23.5. We then use that URL for two things. We use an `ImageViewFragment` to display it on the phone and we use the Anymote Service to fling it to the TV as an `Intent`. The flinging occurs in lines 18 and 19.

LISTING 23.5 Facebook Random Image Remote App

```
1:  @Override
2:  public void onCreate(Bundle savedInstanceState) {
3:    super.onCreate(savedInstanceState);
4:    setContentView(R.layout.main);
5:    LoadPhotos lp = new LoadPhotos(“99394368305” );
6:    lp.execute();
7:    progressBar = (ProgressBar) findViewById(R.id.progressBar1);
8:    progressBar.setVisibility(View.VISIBLE);
9:    mButton2= (Button)findViewById(R.id.button2);
10:   mButton2.setEnabled(false);
11:   mButton2.setOnClickListener(new OnClickListener() {
12:     public void onClick(View v) {
13:       ImageViewFragment fbImg = new ImageViewFragment();
14:       Bundle args = new Bundle();
15:       int random = (int)(Math.random() * mPagePhotos.size());
16:       String url = mPagePhotos.get(random).source;
17:       args.putString(“URL”, url);
18:       final Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
19:       anymoteSender.sendUrl (intent.toUri(Intent.URI_INTENT_SCHEME));
20:       fbImg.setArguments(args);
21:       FragmentTransaction ft = getFragmentManager().beginTransaction();
22:       ft.replace(R.id.linearLayout1, fbImg, “Image from Facebook”);
23:       ft.commit();
24:     }
25:   });
26:   handler = new Handler();
```

```
  Intent intent = new Intent(ExampleTwoActivity.this, AnymoteClientService.class);
  bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
}
```

Both of the examples used simple interfaces on the remote device and displayed the resulting content on the TV. They show the basic use of flinging a URL to the TV for display in a Chrome browser. It is easy to envision more complex user interfaces that also result in a URL being displayed on the TV. For example, an app could allow a user to enter complex search criteria into a phone and have the result displayed on the TV.

Sending KeyEvents to the TV

We'll continue to follow the basic structure of a remote app. The Anymote protocol can handle sending Android Intents, KeyEvents, and mouse events. Using KeyEvents, an app can send whatever keystrokes are entered on a remote device directly to the Google TV app that is being displayed. It can be used to enter search terms on YouTube or other apps, and it is a way to fill out a web form being displayed in the Google Chrome browser.

Handling onKeyDown and onKeyUp Methods

By implementing an Activity's `onKeyDown` and `onKeyUp` methods for the Activity, we can correctly send KeyEvents to the TV using the Anymote protocol. We'll do that in an Activity called `Hour23EchoRemote`. Listing 23.6 shows the `onKeyDown` and `onKeyUp` methods.

LISTING 23.6 onKeyDown and onKeyUp Methods

```
1:  @Override
2:    public boolean onKeyDown(int keyCode, KeyEvent event) {
3:      anymoteSender.sendKey(Code.valueOf(keyCode),Key.Action.DOWN);
4:      return true;
5:    }
6:    @Override
7:      public boolean onKeyUp(int keyCode, KeyEvent event) {
8:        anymoteSender.sendKey(Code.valueOf(keyCode),Key.Action.UP);
9:        return true;
10:   }
```

Listing 23.6 used the `anymoteSender.sendKey` method in lines 3 and 8. The `sendKey` method takes a `Code` and an `Action` as parameters. The `keyCode` parameter passed to the `onKeyUp` and `onKeyDown` events is an `int`. That `int` corresponds to the physical key pressed. The `sendKey` method requires a `Code` corresponding to that key. The `Code` class was defined in the `keycodes.proto` file. By using `Code.valueOf(keyCode)` in line 8, we get the desired result.

The `Action` represents either a key up or key down event.

By explicitly using these KeyEvents, we are sending the TV precisely what was entered by pressing a key on the remote device. If the user presses the special keys on the phone to shift to capital letters, numbers, or special characters, that is reflected in what is sent to the TV.

The Blackjack app used a different method. When the user chose Hit or Stand, the H or S key was sent using the `anymoteSender.sendKeyPress` method:

```
anymoteSender.sendKeyPress(keyEvent);
```

Where `keyEvent` is either `KeyEvent.KEYCODE_H` or `KeyEvent.KEYCODE_S`. The `sendKeyPress` method always sends the value of the physical key from the keyboard. It will send a capital H regardless of whether lowercase h or uppercase H was selected on the device.

For this example app, we implemented the `onKeyUp` and `onKeyDown` methods for the Activity itself. There was no UI with `EditText` fields to accept the input. If there had been, the Activity would never have received the keystrokes. The `EditText` would have consumed them.

To show the keyboard on the Activity, we use the following setting in the Android Manifest for this project:

```
android:windowSoftInputMode="stateVisible"
```

The method `setOnKeyListener (View.OnKeyListener l)` is available for all Views. An alternative to showing an empty Activity with an onscreen keyboard for input is to create a UI that uses Views like `EditText` fields to send keystrokes to the TV. To do that, the `EditText` field would use the `setOnKeyListener` method.

The `onKeyListener` requires the `onKey` method to be implemented. That method passes the `keyCode` and `KeyEvent`, so it would be straightforward to implement an input field for accepting keystrokes and sending them to the TV. The definition of the `onKey` method looks like this:

```
public abstract boolean onKey (View v, int keyCode, KeyEvent event)
```

Developing the TV App

We can use the `Hour23EchoRemote` app as the basis for creating an interactive app. We defined the remote app. Now we need to create the companion app that runs on the TV. Our remote app just sends keystrokes to the TV, so our companion app on the TV will be very simple. We'll show an `EditText` field that can be filled out.

Listing 23.7 shows the entire app that runs on the TV. The layout file contains an `EditText` field to display input. When a key is pressed in the remote app, it is sent to the companion app and displayed. Uppercase, lowercase, and special characters are sent properly from the remote to the TV. To tie these apps together, we must modify the remote code to launch this app on the TV when pairing is complete.

LISTING 23.7 Companion Activity on TV for Echoing Keystrokes

```
public class Hour23Echo extends Activity {
  EditText echoText;
  @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      echoText= (EditText)findViewById(R.id.echo);
  }
}
```

Our goal is for the companion app to launch on the TV when the remote app successfully pairs with the TV. We can look to the Blackjack app as a model for how to do this. We need to fling the Intent we want to start on the TV. That is, we want to create an Intent that corresponds to the Activity on the TV and start that Activity.

Listing 23.8 shows the code to start the TV app from the remote app. The action occurs in the `onConnected` method.

The name of the Activity on the TV is `Hour23Echo`, but more specifically, we should refer to the package name and the class name to create the intent. The package name is `com.bffmedia.hour23echo`. In line 5 of Listing 23.8, we create a new Intent named `echoIntent`. Lines 6 to 8 set `echoIntent` to the values for the package and the class. Line 9 sends the Intent from the remote app to the TV. We are creating Android Intent and using the Anymote protocol to request that the TV launch the Intent.

LISTING 23.8 Starting the Companion App from the Remote App

```
1:  @Override
2:  public void onConnected(final AnymoteSender anymoteSender) {
3:    this.anymoteSender = anymoteSender;
4:    //Add this section to open the Companion App
5:    final Intent echoIntent = new Intent(“android.intent.action.MAIN”);
6:    echoIntent.setComponent(new ComponentName(
7:        “com.bffmedia.hour23echo”,
8:        “com.bffmedia.hour23echo.Hour23Echo”));
9:    anymoteSender.sendIntent(echoIntent);
10:   handler.post(new Runnable() {
11:     public void run() {
12:       progressBar.setVisibility(View.INVISIBLE);
13:     }
14:   });
15: }
```

Sending Mouse Events to the TV

Using the Anymote protocol we have the capability to send messages to the TV that appear to the TV as mouse events. That is, the messages are equivalent to using the mousepad or trackpad on the remote that comes with the TV. We do this using the Anymote Library `sendMoveRelative` method.

Using the Accelerometer

For this remote app, we will use the phone's `Accelerometer` to create the values to send to the TV. Android phones have sensors that can detect a number of values from the real world. The `Accelerometer` detects how the phone is accelerating through space. It detects three values as the phone moves. The x, y, and z values represent a three-dimensional coordinate system, as shown in Figure 23.2.

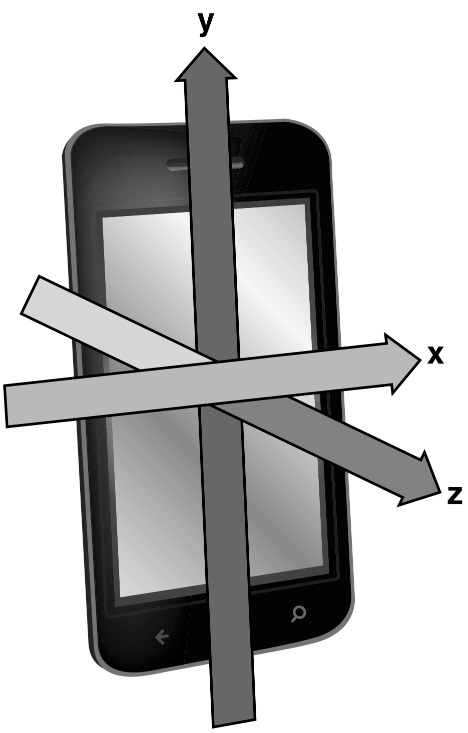

FIGURE 23.2
The coordinate system. Image from Google.

If you move the phone to the right, the x value increases. If you push the phone straight away from you, the y value increases. If you move the phone upward, the z value increases.

We will do a simple mapping of the returned x and y values to create mouselike movements to send to the TV. The effect is to use the phone as a mouse where tilting and moving results in the mouse cursor moving in a reasonable way on the TV.

We implement one `Button` in the app. It is a Click Button that sends the equivalent of a mouse click to the TV. This makes our remote useful as a mouse, but does not provide for D-Pad navigation, a back button, or any other keyboard input.

Listing 23.9 shows the `onCreate` method for the `Accelerometer` remote app. This handles the Click Button by sending mouse down and mouse up keys to the TV.

LISTING 23.9 Accelerometer App onCreate

```
Button click = (Button) findViewById(R.id.button1);
click.setOnClickListener(new OnClickListener() {
  @Override
  public void onClick(View v) {
    anymoteSender.sendKey (Code.BTN_MOUSE, Action.DOWN);
    anymoteSender.sendKey (Code.BTN_MOUSE, Action.UP);
  }
});
```

Detecting Changes Using SensorEventListener

To use the Accelerometer, we must implement a `SensorEventListener` for the Activity:

```
public class Hour23AccelerometerRemote extends Activity implements ClientListener,
SensorEventListener{
```

A `SensorManager` field is defined for the class:

```
private SensorManager sensorManager;
```

The logic for the Accelerometer remote app is the following:

1. Pair with the TV.
2. Begin listening for Accelerometer events.
3. Send those events to the TV as mouse movements.

To accomplish this, we create the `SensorManager` after the connection is made between the TV and the remote, so we do it in the `onConnected` method. The `onConnected` method for the example is shown in Listing 23.10.

LISTING 23.10 Defining SensorManager in onConnected Method

```
1:  @Override
2:  public void onConnected(final AnymoteSender anymoteSender) {
3:    this.anymoteSender = anymoteSender;
4:    sensorManager=(SensorManager)getSystemService(SENSOR_SERVICE);
5:    sensorManager.registerListener(this,
6:    sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
7:        SensorManager.SENSOR_DELAY_FASTEST);
8:    handler.post(new Runnable() {
9:      public void run() {
10:        progressBar.setVisibility(View.INVISIBLE);
11:      }
12:   });
13: }
```

The sensorManager is defined in line 4 of Listing 23.10. In lines 5 to 8, the sensorManager is set up to listen for Accelerometer events. We use the flag SensorManager.SENSOR_DELAY_FASTEST to capture all values from the sensor.

For the SensorEventListener we must implement two methods: onAccuracyChanged and onSensorChanged. To demonstrate sending a mouse movement to the TV, the work will be done in the onSensorChanged method.

Listing 23.11 shows both methods. Line 6 checks to see if the sensor is an Accelerometer. If it is, the values for x, y, and z are populated. Line 10 sends the mouse movement to the TV. In this implementation, 1 is subtracted from y as a simple way to compensate for gravity. Each value is multiplied by -1 to set the proper direction on the TV screen.

This is a very simple movement detection and translation scheme that handles gravity and shows how to make the connection between a sensor on the phone and an action on the TV.

LISTING 23.11 Detecting Sensor Changes

```
1:  @Override
2:  public void onAccuracyChanged(Sensor Sensor, int accuracy) {
3:  }
4:  @Override
5:  public void onSensorChanged(SensorEvent event) {
6:    if(event.sensor.getType()==Sensor.TYPE_ACCELEROMETER){
7:      float x=event.values[0];
8:      float y=event.values[1];
9:      float z=event.values[2];
10:     anymoteSender.sendMoveRelative((int)x*-1,(int) ((y-1)*-1));
11:   }
12: }
```

Using the Accelerometer, we can detect motion of a phone and successfully translate that motion into mouse events on a Google TV. There are a number of other sensors available in most Android phones. Sensors can detect things like temperature, light, and proximity. Using phone sensors and Google TV creates new opportunities for detecting information using a phone and displaying it on a TV.

Summary

In this hour, we created several remote apps using the Anymote library. The basic structure of a remote app was examined. A remote app will implement a `ClientListener` and bind to the `AnymoteClientService`. These apps started the Chrome browser and showed a specific URL by flinging an Intent to the TV. They sent both keystrokes and mouse movements to the TV. Together these example apps provide a basis for creating more sophisticated remote apps.

Q&A

Q. What interface must be implemented by an Activity to use the Anymote protocol?

A. `AnymoteClientService ClientListener` must be implemented. The Activity then acts as a ClientListener and detects when Anymote is connected and disconnected.

Q. Two actions that can be sent from an Activity to a TV mimic what can be done on the TV remote keyboard. What are they?

A. Sending a keystroke via `SendKey` mimics entering a keystroke on the remote control. It is used as follows:

```
anymoteSender.sendKey (Code.BTN_MOUSE, Action.DOWN);
```

Sending a move event mimics a mouse movement on the remote. It is used as follows:

```
anymoteSender.sendMoveRelative(x,y);
```

Q. How can a sensor on a phone like the accelerometer be used in second screen apps ?

A. At a high level, anything detected by the sensor can be communicated to the TV using the Anymote protocol and commands to fling, send keystrokes, and move the mouse. The sensor detects something and the app with the sensor translates the results to something that can be sent with the Anymote protocol to communicate the result to the TV.

Workshop

Quiz

1. What method must be implemented in an `onKeyListener`?
2. What interface includes the `onConnected` method?
3. What does fling mean when using the Anymote protocol?
4. How does the remote app start an Activity on the TV?

Answers

1. The `onKey` method must be implemented. That method accepts the `keyCode` and `KeyEvent`, so entered keystrokes can be detected.
2. ClientListeners must implement three methods, including `onConnected`. The others are `onDisconnected` and `onConnectionFailed`.
3. Fling refers to sending an Intent from the remote to the TV. The app can fling a URL to be displayed on the Chrome browser. The Intent is to view the URL.
4. The remote app starts an Activity on the TV by creating an Intent on the remote that refers to the Activity on the TV. The remote app then flings that Intent to the TV using the sendIntent method. For example:

   ```
   anymoteSender.sendIntent(echoIntent);
   ```

Exercises

1. Try the Accelerometer app directly on a Google TV. You can try this at a department store if you don't have your own Google TV.
2. Modify the Accelerometer code to use the x and z values for changing mouse motion on the TV. See what happens.

This page intentionally left blank

Index

A

Accelerometers, 346-347
- phones, incorporating, 316-318

ActionBars, 149, 156-161
- ActionItems, adding, 153-154
- apps
  - adding to, 282-285
  - organizing, 160
- creating, with two tabs, 174
- defining, 283-284
- Fragments, 174
- implementing, 149-150
- LeftNavBar, 155-156
- MenuItems
  - adding, 153-154
  - adding actions to, 158-159
- tabs
  - adding, 151-152
  - adding actions to, 159

ActionItems
- ActionBars, adding to, 153-154

actions
- buttons, Closure UI Library, 70
- Intents, 293
- MenuItems, adding to, 158-159
- tabs, adding to, 159

Activities
- AsyncTask class, 218
- Channel Listing providers, 303-304
- enhancing, 281-282
- Fragments
  - swapping in, 166-173
  - visibility, 171-173
- life cycles, 163-165

ADT (Android Development Tools), 98

AJAX, retrieving videos, 41

All Apps icon, 2

alpha values, ImageViews, setting, 138-139

Android, 112
- ADT (Android Development Tools), 98

apps, 7-8
testing, 100-103
development environment, 97-98
Eclipse, setting up projects, 104-111
ecosystem, 96-97
Emulator, 100-101
history, 96
Honeycomb, 95
operating system, 95-97
SDK (Software Development Kit), 95, 97-98

AndroidManifest.xml file, 289-292, 294, 296, 298
including services and receivers to, 291-292
Intents, 292-294
market filters, 294-295
universal apps, 295

Anymote protocol, 319, 353-358, 364
Chrome extensions, 360-363
defining
keycodes, 353-354
request and response messages, 354-356
library code, exploring, 332-333

APIs (application programming interfaces)
calls, making, 213
NPAPI (Netscape Plug-in API), 360-363

apps, 6
ActionBars, adding, 282-285
Activities
enhancing, 281-282
Fragments, 166-173
life cycles, 163-165
Android, 7-8
testing, 100-103
developing, 269-273
GridFragments, 274-280
planning, 272-273
Netflix, 3
organizing, ActionBars, 160
publishing, 296-297
running multiple versions, 296
second screen, 8, 315-316, 323, 325
Anymote library code, 332-333
connecting TVs, 318-322
content selection, 316
developing, 337, 343-345
enhanced viewing, 316
exploring code, 330-332
flinging URLs, 337-343
interactive, 316
preparing, 325-328
remote control, 316
running, 328-329
sending KeyEvents to TV, 343-345
sending mouse events to TV, 346-349
user interaction, 316-318
TV & Movies, 2
web, 6-7
Closure UI Library, 61-62
jQuery GTV, 53-54
playing video, 20-22
wireframes, creating for, 270
YouTube, 3

AsyncTask, 215, 218
background downloads, 209-210

AsyncTaskLoader, 258

attributes, buttons, 131-132
changing through code, 132-133

Audio Focus, Channel Listing providers, 310-312

AutoCompleteView, 135-136

Autozoom, 15-16
adjusting ratio, 19-20

B

Back button, Fragments, 169-170

background downloads, AsyncTask, 209-210

bitmaps
drawing directly, ImageViews, 139-140
remote images, getting from, 208-209

Blackjack second screen app, 334
Anymote library code, exploring, 332-333
developing, 337, 343-345
flinging URLs, 337-343
exploring code, 330-332
KeyEvents, sending to TV, 343-345
mouse events, sending to TV, 346-349
preparing, 325-328
running, 328-329

Breaking Bad second screen app, 315

Buddy Boxes, 5-6

BuilderPhotoPage (jQuery GTV UI library), 53

BuilderVideoPage (jQuery GTV UI library), 53

building, ContentProviders, 240-247

bundles, passing data, 185-186

Button class (Closure UI Library), 67-69

button overlays, videos, 142-144

buttons
- attributes, 131-132
  - changing through code, 132-133
- Back, Fragments, 169-170
- hover states, 26
- padding, 119
- placing, 124-125
- stylesheets, 50-51

C

calls, API, making, 213

categories, Intents, 293

Channel Listing providers, 301, 313
- Activities, 303-304
- Audio Focus, 310-312
- changing channels, 307-309
- getting data from, 301-304
- listing all channels, 305-306

channels
- changing, Channel Listing providers, 307-309
- listing all, Channel Listing providers, 305-306

Chrome, 3, 6-7
- extensions, Anymote and Pairing protocols, 360-363

classes
- Closure UI Library, 66-67
  - Button, 67-69
  - Component, 71
  - Containers, 72-73
  - Grid, 73-75
  - Lightbox, 75-76
  - Link, 73
  - TabContainer, 77-79
- HTTPUrlConnection, retrieving data, 214-215
- Loaders, 258
- PhotoAdapterViewFragment, 279-280
- PhotoGridCursorAdapter, 276-279
- SQLiteOpenHelper, 226-228
- UrlConnection, 208

Closure UI Library, 61, 80
- classes, 66-67
  - Button, 67-69
  - Component, 71
  - Containers, 72-73
  - Grid, 73-75
  - Lightbox, 75-76
  - Link, 73
  - TabContainer, 77-79
- dependency files, creating, 63-64
- HTML pages, creating, 63-66
- web apps, 61-62

code, Blackjack second screen app, exploring, 330-332

code listings
- Accelerometer App onCreate (23.9), 347
- Accessing Data in the Channel Provider (20.1), 301-304
- Activity Code for Swapping Fragments (11.4), 168-169
- Activity Method to Show a Fragment (11.7), 172
- Activity with OnKeyDown Implemented (13.4), 202-203
- Adding a Listener Interface to a DialogFragment (12.3), 184
- Adding an Adapter in onActivityCreated (18.3), 275
- Adding Buttons with Actions to a Fragment View (11.5), 170
- Adding Drawables to a Button (9.2), 131
- Adding Style to a TextView (9.1), 130
- Adding Tabs to the ActionBar (10.2), 151-152
- Adding the MenuItems (10.7), 158
- Adding the TabListener (10.8), 159
- Adding Unique Fonts (2.9), 24-25
- Adjusting Image Height for 720p or 1080p (2.3), 19
- Adjusting the Autozoom Ratio (2.4), 19
- Adjusting Video Height for 720p or 1080p (2.6), 21-22

Android Manifest for Remote App (22.4), 331
Android Manifest for Remote App (22.5), 333
AndroidManifest.xml (19.1), 289-290
Applying the Style (4.5), 51-52
Assigning a Video to a VideoView (9.13), 141
AsyncTask in Activity (14.11), 218
AutoCompleteTextView App Code (9.6), 135
AutoCompleteView Layout (9.5), 135
Basic Web Sitemap for a Single URL (6.1), 84
Blackjack “Hit” Button Code (22.1), 330
Button Actions (5.6), 70
Button Components (5.5), 69
Button Includes (5.3), 67
Button onClickListener for Random Photo (14.13), 220
Button Styles (5.4), 68-69
Caching an Image for Performance (14.4), 211-212
Calling an Activity Method from a Fragment (11.8), 173
Calling the AsyncTask for Background Image Loading (14.3), 210
Calling the Listener Ready Method (12.4), 184-185
Challenge Option Message (24.5), 360
Changing Video Position from SeekBar (9.15), 143
ChannelGridCursorAdapter Snippet (20.4), 309
Channels in a ListFragment (20.2), 305-306
Checking the Database for Photos (15.12), 234
Cleaning Up in the onDestroy Method (23.4), 341
Coding an Activity Using the Layout (8.2), 116
Companion Activity on TV for Echoing Keystrokes (23.7), 337
ContentObserver Defined in ImageViewFragment (16.15), 253-254
Creating a Bitmap from a URL (14.1), 208
Creating a Bitmap Through Java Code (9.11), 140
Creating a Bitmap Using the ContentProvider (16.14), 252-253
Creating a Data Object with Categories and Videos (3.5), 39-40
Creating a DialogFragment Class (12.1), 182
Creating a Gallery Fragment (12.13), 191
Creating a GridView Fragment (12.11), 189-190
Creating a Listener in the Activity (12.5), 185
Creating an ActionBar with Two Tabs (11.9), 174
Creating an Activity with LeftNavBar, MenuItems, and an ActionItem (10.6), 156-157
Creating the ActionBar (10.1), 150
Creating the GridView (18.1), 274
Creating the Photo Database (15.2), 227
Creating the PhotoCursorGridAdapter (18.4), 276
Creating Video Objects (3.4), 39
Data.html to Dump Data from dataprovider.js (3.3), 37
Declaring the MenuItems (10.3), 153
Defining a Click Method (11.13), 177
Defining a Font-Face (2.8), 24
Defining a ListFragment Class (12.8), 187
Defining a ProgressBar (9.7), 137
Defining an ActionItem (10.5), 154
Defining Component Styles (5.7), 71
Defining MenuItems (10.4), 153
Defining SensorManager in onConnected Method (23.10), 348
Defining the ActionBar (18.12), 283
Defining the Photo Columns (15.8), 232
Delete Method (16.8), 247
Detecting Sensor Changes (23.11), 348

Displaying a ListFragment Class (12.9), 188
Downloading an Image Using an AsyncTask (14.2), 209-210
EditView Layout (9.4), 135
Extending ImageView (13.3), 201-202
Facebook Random Image Remote App (23.5), 342-343
Fixing the FrameLayout (8.6), 123
Getting Photos from a Database in Specified Order (15.9), 232
Getting Photos from the Database (15.7), 231
Grid Components (5.10), 75
Grid Item Layout (18.5), 276-277
Grid Styles (5.9), 73-74
GridView Layout (18.2), 274
Handling a Change in AudioFocus (20.6), 312
Handling a Key Sent from the Remote App (22.3), 331
Hello-tv.html File Using Google TV Closure UI Library (5.1), 65
Hour7Activity.java (7.1), 108
Hovering on Links and Buttons (2.10), 26
HTML to Display a Single Image (2.1), 17
HTML to Display a Single Image with Autozoom Off (2.2), 17
HTML Video Tag (2.5), 21
ImageViewFragment (14.14), 220
ImageViewFragment Loader Setup (17.1), 261
ImageViewFragment Using the Data (17.2), 262-263
Implementing ActionBar. TabListener (11.10), 174-175
Implementing File Support in PhotoProvider (16.12), 250-251
Implementing Rewind (4.2), 48-49
Incrementing the SeekBar with Video Progress (9.14), 143
Index.js Reading from Data Provider (3.2), 37
Index.js Starting the Template Page (3.1), 36
Indicating Episodes in a Video Sitemap (6.4), 90
Inflating a View (FragmentOne. java) {11.3}, 167
Insert Method (16.6), 247
Inserting and Updating Using ContentProvider (16.11), 249
Interacting with Video (4.1), 47
Interpreting a Request Message (24.3), 357
JavaScript for Row Control (4.7), 57-58
JSON Example (14.6), 215
JSONArray Structure (14.7), 216
Keycode Proto File Snippet (24.1), 354
Layout for a Gallery (gallery_ fragment.xml) {12.12}, 191
Layout for a GridView (grid_ fragment.xml) {12.10}, 189
Layout for Fragment (one_frag- ment.xml) {11.2}, 167
Layout Including DpadImageView (13.5), 203
Layout view_fragment.xml (18.9), 280
Layout view_fragment.xml (18.10), 281
Lightbox Style Components (5.11), 75
LinearLayout in XML (8.1), 114
ListDialogFragment onActivity- Created (18.14), 284-285
ListDialogFragment onCreate View (18.13), 284
Loading and Reading a JSONArray (14.8), 217
Main.xml (7.2), 108
Make an HttpUrlConnection Request and Read the Response (14.5), 214
Making a Photo Array from a String (14.10), 218
Making a Photo Object from a JSONObject (14.9), 217
New Methods for the Activity (11.6)l, 172
onFocus Function (5.2), 66
onKeyDown and onKeyUp Methods (23.6), 343

OnPreExecute (16.9), 248

Open, Close, and Update (15.3), 228

Opening a DialogFragment (12.2), 183

PairingRequest and PairingRequestAck (24.4), 359-360

Passing Data in the Activity (12.6), 186

PhotoAdapterViewFragment onLoadFinished (18.8), 279

PhotoCursorGridAdapter BindView (18.6), 277-278

PhotoCursorGridAdapter BindView (18.7), 278-279

PhotoProvider Declarations (16.2), 242

PhotoProvider Query Method (16.3), 243

Placing Buttons with RelativeLayout (8.7), 124-125

Price and Subscription Options (6.3), 88

Progress Bar (4.3), 50

Random Result from Cursor (16.10), 248

Read Data in the Fragment (12.7), 186

Remote Activity onCreate Method (23.2), 340

Request Messages in the Remote Proto File (24.2), 355

Requesting AudioFocus (20.5), 311

Required Methods for ClientListener (23.3), 341

Retrieving an Image in PhotoProvider (16.13), 251-252

Retrieving Data in Background to Populate Photo Array (14.12), 219

Return Type of Data in GetType (16.5), 246

Returning a Photo Object from a Cursor (15.10), 233

Rewinding a Video (9.16), 144

Sending a Key to TV App (22.2), 331

ServiceConnection Declaration (23.1), 339

Setting Alpha Value Through Java Code (9.10), 139

Setting Button Padding (8.4), 120

Setting Navigation Focus in an XML Layout (13.1), 197-198

Setting ScaleType to centerInside for an ImageView (9.9), 138

Setting ScaleType to fitXY for an ImageView (9.8), 137

Shell of a ContentProvider (16.1), 241

Showing Activity Life Cycle (11.1), 164

Showing Photo Data in a List with PhotoListFragment.java (17.3), 263-264

Showing the Lightbox Using JavaScript (5.12), 76

ShowPagePhoto Method (18.11), 282

Starting the Companion App from the Remote App (23.8), 345

Strings.xml (7.3), 109

Styles for Row Control (4.6), 56

Stylesheet for Buttons: video-styles.css (4.4), 51

Supporting Another URI in Query (16.4), 245

Supporting Play and Pause on a Remote (2.11), 26-27

Switch Based on KeyCode (13.2), 200

TabContainer HTML for Tab Content (5.14), 77

TabContainer HTML for Tabs (5.13), 77

TabContainer Styles (5.15), 78-79

TabListener onTabSelected Method (11.11), 176

TabListener onTabUnselected Method (11.12), 176-177

Text Size and Color (2.7), 23

Update Method (16.7), 247

Updating Button Attributes Within Code (9.3), 133

Updating the Database from an ArrayList of Photo Objects (15.11), 234

Updating the LinearLayout XML file (8.3), 117

Using a Cursor to Select a Random Photo (15.13), 235

Using Component Styles (5.8), 72

Using SQLiteOpenHelper (15.1), 226

Using the Click Method Within Activity (11.14), 177

Video Sitemap with Required Elements (6.2), 87
VideoView Layout (9.12), 140
color, displaying, 22-23
Component class (Closure UI Library), 71
components
buttons, Closure UI Library, 69
lightboxes, Closure UI Library, 76
Containers class (Closure UI Library), 72-73
content selection apps, 316
ContentProviders, 239, 255
building, 240-247
declarations, 241-243
delete method, 247
GetType method, 246
insert method, 247
PhotoProvider, 248-249
query method, 243-245
requesting files from, 249-254
required methods, 240-241
retrieving images, 251-253
returning files from, 250-251
shell, 241
update method, 247
URIs (Uniform Resource Identifiers), data retrieval, 239-240
controls
jQuery GTV UI library, 53-54
RowControl, 55-57
video, adding, 46-50
Controls property (HTML5MediaElement), 46
CreationParams, jQuery GTV UI library, 55
CurrentTime property (HTML5MediaElement), 46
CursorAdaptors, CursorLoaders, 263-265
CursorLoaders, 257-258, 266
CursorAdaptors, 263-265
images, displaying, 261-263

D

Dalvik Debug Monitor Server (DDMS), 103
data, Channel Listing providers, getting from, 301-304
data objects, creating with categories and videos, 39-40
data retrieval, URIs (Uniform Resource Identifiers), 239-240
databases
organizing with tables, SQLite, 225-226
photo objects, 233-234
dataprovider.js file, retrieving videos, AJAX, 41
Dataprovider.js file, Template 2, 34-35, 37-38
DDMS (Dalvik Debug Monitor Server), 103
declarations, ContentProviders, 241-243
defining, ActionBars, 283-284
deleting photographs, SQLite, 230
dependency files, creating, Closure UI Library, 63-64
desktop computers versus Google TV, 4
developing, GridFragments, 274-280
developing apps, 269-273
planning, 272-273
development environment, Android, 97-98
devices, working on, 101-103
DialogFragments, 181-186, 191
getting data from, 183-185
opening and closing, 181-183
displaying
color, 22-23
remote images, 207-212
text, 22-25
videos, VideoViews, 212
displays versus Google TV, 4
doInitializationPhase() method, 333
D-Pad, 2, 195, 205
designing for, 195-196
key events, programming, 199-203
navigating with, 196
remote controls, 195
duration attribute, Video Sitemaps, 91

E

Eclipse
Android, 98
projects, 327-328
setting up, 104-111
ecosystems, Android, 96-97
EditView, 134-135
Emulator (Android), 100-101

enhanced viewing apps, 316
episode tags, Video Sitemaps, 89-90
events, 134
  D-Pad, programming key, 199-203
extensions, Chrome, Anymote and Pairing protocols, 360-363

F

family friendly attribute, Video Sitemaps, 91
fast forward control, implementing, 48-49
fetching remote data, 212-215
files, AndroidManifest.xml, 289-292, 294, 296, 298
  Intents, 292-294
  market filters, 294-295
  universal apps, 295
filters, market, AndroidManifest.xml file, 294-295
focus, navigation, 197-198
folders, Closure UI Library, 62
fonts, 23-25
  true type face, 24
  unique, adding, 24-25
form widgets, 129
  AutoCompleteView, 134-136
  buttons, 131-134
  EditView, 134-135
  ProgressBars, 136
  TextViews, styling, 129-131
formatted data, JSON (JavaScript Object Notation), 217-218
Fragments, 165-166, 178
  ActionBars, 174-177
  Activities
    swapping in, 166-173
    visibility, 171-173
  Back button, 169-170
  creating
    Gallery, 190-191
    GridViews, 188-190
  GridFragments, 272-273
    Developing, 274-280
  Java classes, creating, 167-169
  ListDialogFragment, 284-285
  specialized, 181, 193
    DialogFragments, 181-186, 191
    ListFragments, 186-188
  TextViews, adding to, 171
  UI components, adding to, 170-171
  XML layouts, 166-167
FragmentTransation parameter, 175-177
FrameLayout, 122-123, 142

G

Gallery, Fragments, creating, 190-191
games, WeDraw, 10
GetGlue, 315
GetType method, ContentProviders, 246
Google Play, 3
Google TV, capabilities, 3-6
Google Webmaster Tools, 84-85
  Video Sitemaps, 91
Grid class (Closure UI Library), 73-75
GridFragments, 272-273
  developing, 274-280
GridViews, Fragments, creating, 188-190

H

Height, video, adjusting, 21-22
Honeycomb, 95
hover states, buttons and links, 26
HTML (HyperText Markup Language)
  displaying images on full screen, 17-19
  IMG tag, 16
  pages, creating, 63-66
HTML 5, 45
  video tag, 21, 45-46
  video templates, 31
    features, 33
    handling fewer videos on single page, 32
    handling many videos on separate pages, 32
    showing videos, 38-40
    structure, 33-34
    Template 1, 32
    Template 2, 32-33, 36-38
HTML5MediaElement, 46
HTTPUrlConnection class, retrieving data, 214-215

I

icons
- All Apps, 2
- Google Play, 3
- Live TV, 2
- Search, 3

images. See also photographs
- caching locally, 211-212
- displaying on full screen, 16-19
- remote, displaying, 207-212

ImageViewFragment, Loaders, adding to, 260-265

ImageViews, 137-140
- drawing bitmaps directly, 139-140
- extending, 201-202
- ScaleType property, 137-139
- setting alpha value, 138-139

IMG HTML tag, 16

Index.js file, Template 2, 34-37

initialization, Loaders, 259

installing, Chrome extensions, 360

Intents, AndroidManifest.xml file, 292-294

interactive apps, 316

interface. See UI (user interface)

J

JavaScript, Template 2 (video), 36-38

jQuery GTV UI library, 45, 52-57
- controls, 53-54
  - RowControl, 55-57
- web apps, running, 53-54

JSON (JavaScript Object Notation)
- formatted data, 215-218
- JSONArray, 216-217
- JSONObject
  - creating, 216
  - making photo object from, 217
- parsing, 217-218

JSONArray, 216-217

JSONObject
- creating, 216
- making photo object from, 217

K-L

key events, D-Pad, programming, 199-203

keycodes, defining, Anymote protocol, 353-354

laying out, widgets, 119-121

layouts, 113-114
- FrameLayout, 122-123
- LinearLayout, 113-118
  - XML, 114-115
- margins, 121
- RelativeLayout, 124-126
- XML, Fragments, 166-167

Leanback, 10-11

LeftNavBar, ActionBars, 155-156

libraries
- Closure UI Library, 61, 80
  - classes, 66-79
  - dependency files, creating, 63-64
  - HTML pages, creating, 63-66
  - web apps, 61-62
- jQuery GTV UI library, 45, 52-57
  - controls, 53-57
  - web apps, running, 53-54

life cycles, Activities, 163-165

Lightbox class (Closure UI Library), 75-76

LinearLayout, 113-118
- XML, 114-115

Link class (Closure UI Library), 73

links, hover states, 26

ListDialogFragment, 284-285

listeners, 134
- VideoViews, 141

ListFragments, 186-188

listings. See code listings

live streaming, Video Sitemaps, 90-91

Live TV icon, 2

LoaderManager, 258

LoaderManagerCallbacks, 258

Loaders, 257
- classes, 258
- creating, 259
- ImageViewFragment, adding to, 260-265
- initializing, 259
- resetting, 260
- states, 258-260

loading videos, VideoViews, 140-141

Logcat, Activities, checking results, 165

Logitech Revue, 27

Loop property (HTML5MediaElement), 46

How can we make this index more useful? Email us at indexes@samspublishing.com

M

manifest file, 289-292, 294, 296, 298
- including services and receivers to, 291-292
- Intents, 292-294
- market filters, 294-295
- universal apps, 295

margins, layouts, 121

market filters, AndroidManifest. xml file, 294-295

menu tray, 2-3

MenuItems, ActionBars
- adding actions to, 158-159
- adding to, 153-154

messages
- request
  - defining, 354-356
  - interpreting, 357
- response, defining, 354-356

methods
- doInitializationPhase(), 333
- GetType, ContentProviders, 246
- newTab(), 152
- onCreate, 340
- onDestroy, 341
- onKeyDown, 199-200, 343-344
- onKeyUp, 343-344
- sendKeyEvent, 331

mobile phones versus Google TV, 4

mouse events, sending to TV, 346-349

Muted property (HTML5MediaElement), 46

N

navigation, 25
- focus, 197-198
- making obvious, 25-26
- remote control, supporting, 26-27

navigation bar, search results, 84

Netflix app, 3

Netscape Plug-in API (NPAPI), 360-363

newTab() method, 152

notifications, 2

NPAPI (Netscape Plug-in API), 360-363

O

objects
- data, creating, 39-40
- video, creating, 39

onCreate method, 340

onDestroy method, 341

onFocus function, 66

onKeyDown method, 199-200, 343-344

organizing apps, ActionBars, 160

overscan, 16

P

padding, buttons, 119

Pairing protocol, 320, 353, 358-360, 364
- Chrome extensions, 360-363

parsing, JSON (JavaScript Object Notation), 217-218

passing data, bundles, 185-186

PhotoAdapterViewFragment class, 279-280

PhotoControl (jQuery GTV UI library), 53

photographs
- deleting, SQLite, 230
- inserting, SQLite, 228-229
- updating, SQLite, 229-230

PhotoGridCursorAdapter class, 276-279

PhotoProvider, 248-249

PlaybackRate property (HTML5MediaElement), 46

playing video, 20-22, 144

Poster property (HTML5MediaElement), 46

preparing, second screen apps, 325-328

price attribute, Video Sitemaps, 88

processors, 4

programming D-Pad key events, 199-203

ProgressBars, 136
- adding, 49-50

projects, Eclipse, setting up, 104-111

protocols
- Anymote, 353-358, 364
  - Chrome extensions, 360-363
- Pairing, 353, 358-360, 364
  - Chrome extensions, 360-363

providers
- Channel Listing providers, 301, 313
  - Activities, 303-304
  - Audio Focus, 310-312
  - changing channels, 307-309
  - getting data from, 301-304
  - listing all channels, 305-306
- ContentProviders, 239, 255
  - building, 240-247
  - declarations, 241-243
  - delete method, 247
  - GetType method, 246
  - insert method, 247
  - PhotoProvider, 248-249
  - query methods, 243-245
  - requesting files from, 249-254
  - required methods, 240-241
  - retrieving images, 251-253
  - returning files from, 250-251
  - shell, 241
  - update method, 247
  - URIs (Uniform Resource Identifiers), data retrieval, 239-240
- PhotoProviders, 248-249

publishing apps, 296-297

Q-R

query methods, ContentProviders, 243-245

querying data, cursors, 231-233

Racebuddy (NASCAR.com), 315

rating attribute, Video Sitemaps, 91

RelativeLayout, 124-126, 142

remote control apps, 316

remote control navigation, supporting, 26-27

remote data, fetching, 212-215

remote images
- bitmaps, getting from, 208-209
- caching locally, 211-212
- displaying, 207-212

request messages
- defining, Anymote protocol, 354-356
- interpreting, 357

resetting, Loaders, 260

resolution, 16

response messages, defining, Anymote protocol, 354-356

restrictions, Video Sitemaps, 89

retrieving videos, AJAX, 41

rewind functionality, implementing, 48-49, 144

RollerControl (jQuery GTV UI library), 53

RotatorControl (jQuery GTV UI library), 53

RowControl (jQuery GTV UI library), 53
- creating, 55-57

running, Blackjack second screen app, 328-329

S

ScaleType property, ImageViews, 137-139

screen overscan, 16

screen resolution, 16

SDK (Software Development Kit), Android, 95, 97-98

Search icon, 3

searches, Video Sitemaps, 84

second screen apps, 8, 315-316, 323, 325
- Anymote library code, exploring, 332-333
- code, exploring, 330-332
- connecting TVs, 318-322
- content selection, 316
- developing, 337, 343-345
  - flinging URLs, 337-343
- enhanced viewing, 316
- interactive, 316
- KeyEvents, sending to TV, 343-345
- mouse events, sending to TV, 346-349
- preparing, 325-328
- remote control, 316

running, 328-329
user interaction, 316-318
SeekBar, 142-144
sendKeyEvent method, 331
SensorEventListener, 347
ServiceConnection declaration, 339
ShowParams, jQuery GTV UI library, 55-56
SideNavControl (jQuery GTV UI library), 53
SlidingControl (jQuery GTV UI library), 53
smart phones versus Google TV, 4
Software Development Kit (SDK), Android, 95, 97-98
specialized Fragments, 181, 193
DialogFragments, 181-186, 191
ListFragments, 186-188
SQLite, 225, 233-237
databases, organizing with tables, 225-226
photographs
deleting, 230
inserting, 228-229
updating, 229-230
querying data, cursors, 231-233
SQLiteOpenHelper class, 226-228
SQLiteOpenHelper class, 226-228
StackControl (jQuery GTV UI library), 53
states, Loaders, 258-260
styles
applying, 51
Closure UI Library
buttons, 68-69
components, 71
grids, 73-74
TabContainer, 78-79
TextViews, adding to, 129-131
stylesheets, buttons, 50-51
swapping, Fragments, Activities, 166-173

T

TabContainer class (Closure UI Library), 77-79
tables, databases, organizing, 225-226
tablets versus Google TV, 4
TabListener, 176-177
tabs, ActionBars
adding actions to, 159
adding to, 151-152
tag attribute, Video Sitemaps, 91
tags
HTML 5 video, 21
Video Sitemaps, 87-91
television screens, displaying content on, 15-19
televisions versus Google TV, 4
templates (video), 31
features, 33
handling fewer videos on single page, 32
handling many videos on separate pages, 32
showing videos, 38-40
structure, 33-34
Template 1, 32
Template 2, 32-33, 36-38
ten-foot user experience, 10-13
designing for, 11-12
Leanback, 10-11
testing Android apps, 100-103
text, displaying, 22-25
TextViews
Fragments, adding to, 171
styling, 129-131
true type face fonts, 24
TV & Movies app, 2
TV displays versus Google TV, 4

U

UI (user interface), 1-3
components, adding to Fragments, 170-171
home screen, 1
menu tray, 2-3
notification indicator, 2
unique fonts, adding, 24-25
universal apps, AndroidManifest.xml file, 295
updating photographs, SQLite, 229-230
URIs (Uniform Resource Identifiers)
data retrieval, ContentProviders, 239-240
forming, data access, 302

UrlConnection class, 208

user interaction, second screen apps, 316-318

V

video controls, adding, 46-50

video ID attribute, Video Sitemaps, 87-88

video objects, creating, 39

Video Sitemaps, 92
- adding details to, 85-87
- creating, 83
- episodes, 89-90
- Google Webmaster Tools, 84-85, 91
- live streaming, 90-91
- price attribute, 88
- restrictions, 89
- searches, 83-84
- tags, 87-91
- video ID attribute, 87-88
- videos, requirements, 86

video templates, 31
- features, 33
- handling fewer videos on single page, 32
- handling many videos on separate pages, 32
- showing videos, 38-40
- structure, 33-34
- Template 1, 32
- Template 2, 32-33
  - JavaScript, 36-38

VideoControl (jQuery GTV UI library), 53

videos
- adjusting height, 21-22
- button overlays, 142-144
- displaying, VideoViews, 212
- loading, VideoViews, 140-141
- playing, 20-22, 144
- retrieving, AJAX, 41
- starting, pausing, and positioning, VideoViews, 141
- Video Sitemap requirements, 86

VideoViews, 140-141
- listeners, 141
- videos
  - displaying, 212
  - loading, 140-141
  - starting, pausing, and positioning, 141

views
- AutoCompleteView, 135-136
- drawings, creating, 200
- EditView, 134-135
- GridViews, 188-190
- ImageViews, 137-140
  - drawing bitmaps directly, 139-140
  - extending, 201-202
  - ScaleType property, 137-139
  - setting alpha value, 138-139
- TextViews
  - Fragments, adding to, 171
  - styling, 129-131
- VideoViews, 140-141

visibility, Activities, Fragments, 171-173

Volume property (HTML5MediaElement), 46

W

web apps, 6-7
- Closure UI Library, 61-62
- jQuery GTV, running, 53-54
- playing video, 20-22

WeDraw, 10

Widgets, 146
- form, 129, 131-134
  - AutoCompleteView, 135-136
  - EditView, 134-135
  - ProgressBars, 136
- Gallery, creating Fragments, 190-191
- laying out, 119-121
- SeekBar, 142-144

wireframes, apps, creating for, 270

X-Y

XML layouts
- Fragments, 166-167
- LinearLayout, 114-115

YouTube app, 3

YouTube Leanback, 10-11

How can we make this index more useful? Email us at indexes@samspublishing.com