Infringement of United States Patent number 7,716,629 by the Samsung's Android-Powered Smartphones

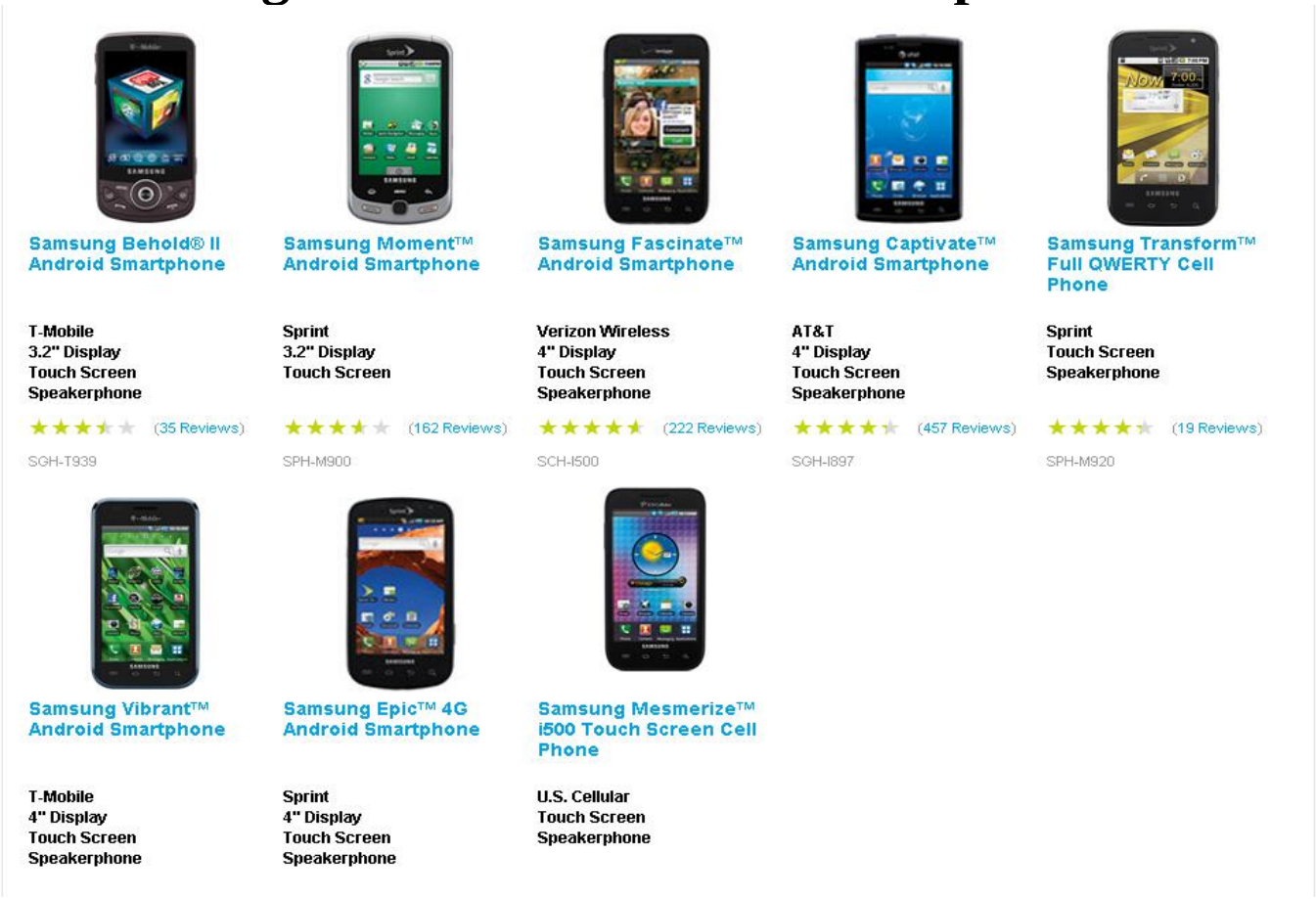


FIGURE 1 (List Samsung's Android-Powered Smartphones from the SAMSUNG.COM Website)

The Samsung Android-Powered Smartphone on Figure 1 infringe claims of U.S. Patent number 7,716,629 because they include "**A system for generating a computer application on a host system in an arbitrary object framework that separates a content of said computer application, a form of said computer application, and a functionality of said computer application.**" Samsung imports these Smartphones into the United States.

A user of these Samsung Android-Powered Smartphones can generate a computer application within the Android framework on Samsung's Android-Powered Smartphones themselves by using text editors or other text manipulation techniques provided on the Smartphones. After compiling the application, the executable can be put back on the Smartphone and run.

Infringement of United States Patent number 7,716,629 by the Samsung's Android-Powered Tablet Computers

All those are Android Powered Tablet Computers.

FIGURE 2 (List Samsung's Android-Powered Tablet Computers from the SAMSUNG.COM Website)

The Samsung Android-Powered Tablet Computers on Figure 2 infringe claims of U.S. Patent number 7,716,629 because they include "**A system for generating a computer application on a host system in an arbitrary object framework that separates a content of said computer application, a form of said computer application, and a functionality of said computer application.**" Samsung imports these Tablet Computers into the United States.

A user of these Samsung Android-Powered Tablet Computers can generate a computer application within the Android framework on Samsung's Android-Powered Tablet Computers themselves by using text editors or other text manipulation techniques provided on the Tablet Computers. After compiling the application, the executable can be put back on the Tablet Computer and run.

We will refer from now on to both Samsung's Android-Powered Smartphones and Samsung's Android-Powered Tablet Computers as "Android Devices".

In order to explain this, and how the Android Framework infringes, the book "Professional Android 2 Application Development", ISBN 978-0-470-5656552-0 written by Reto Meier will be used. "Reto Meier is a software developer who has been involved in Android since its initial release in 2007. He is an Android Developer Advocate at Google". This book will be referred to as the "Meier Book".

Patent # 7,716,629 Claim Chart for the Android Framework
GENERATION OF AN ANDROID APPLICATION ON THE ANDROID DEVICES

The Android Devices are all Android-Powered. The Android Framework includes the Linux Operating System. As per the Meier Book, Page 14, in the lowermost block, it shows that the Linux is the lowest part of the Android Framework:

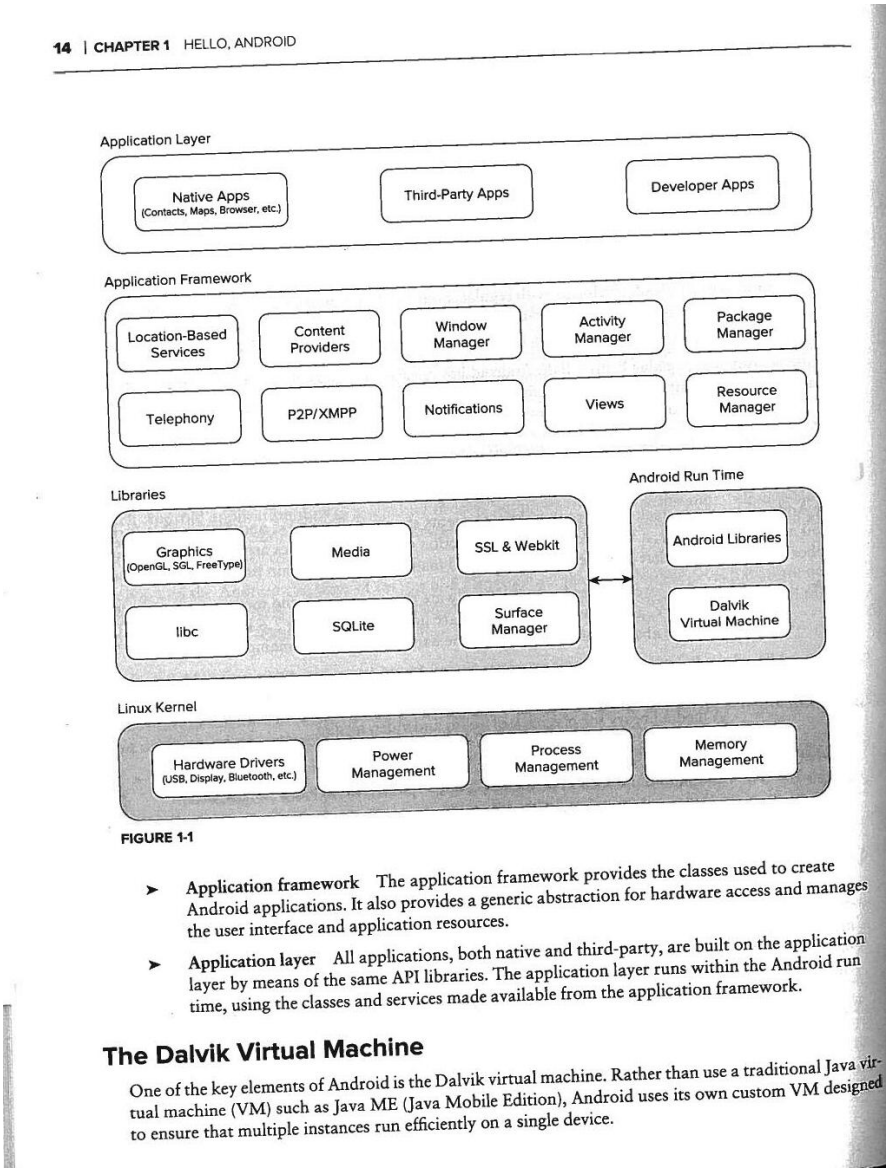
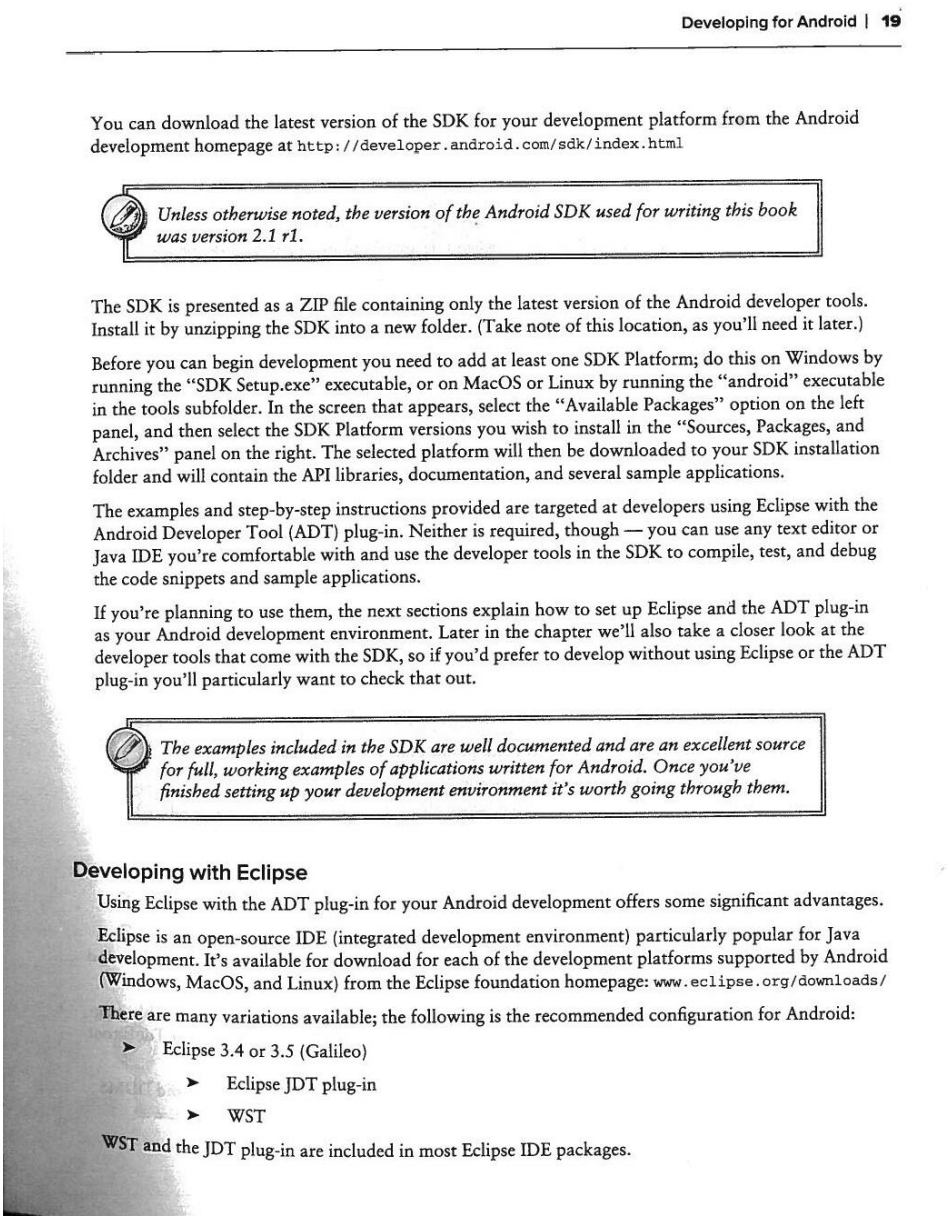


FIGURE 3 (Meier Book, Page 14)

The Linux Portion of the Android Framework includes built-in command line commands such as CAT, VI, CP, MV and RM. (Cat Can create text files, VI can modify them, CP, MV and RM can Copy , Move, Rename and Delete files).

Patent # 7,716,629 Claim Chart for the Android Framework
In The Meier Book, Page 19, 4th paragraph, it shows that to create Android Applications , all that its needed is a Text Editor, or anything that can create text files:



You can download the latest version of the SDK for your development platform from the Android development homepage at `http://developer.android.com/sdk/index.html`

Unless otherwise noted, the version of the Android SDK used for writing this book was version 2.1 r1.

The SDK is presented as a ZIP file containing only the latest version of the Android developer tools. Install it by unzipping the SDK into a new folder. (Take note of this location, as you'll need it later.)

Before you can begin development you need to add at least one SDK Platform; do this on Windows by running the "SDK Setup.exe" executable, or on MacOS or Linux by running the "android" executable in the tools subfolder. In the screen that appears, select the "Available Packages" option on the left panel, and then select the SDK Platform versions you wish to install in the "Sources, Packages, and Archives" panel on the right. The selected platform will then be downloaded to your SDK installation folder and will contain the API libraries, documentation, and several sample applications.

The examples and step-by-step instructions provided are targeted at developers using Eclipse with the Android Developer Tool (ADT) plug-in. Neither is required, though — you can use any text editor or Java IDE you're comfortable with and use the developer tools in the SDK to compile, test, and debug the code snippets and sample applications.

If you're planning to use them, the next sections explain how to set up Eclipse and the ADT plug-in as your Android development environment. Later in the chapter we'll also take a closer look at the developer tools that come with the SDK, so if you'd prefer to develop without using Eclipse or the ADT plug-in you'll particularly want to check that out.

The examples included in the SDK are well documented and are an excellent source for full, working examples of applications written for Android. Once you've finished setting up your development environment it's worth going through them.

Developing with Eclipse

Using Eclipse with the ADT plug-in for your Android development offers some significant advantages.

Eclipse is an open-source IDE (integrated development environment) particularly popular for Java development. It's available for download for each of the development platforms supported by Android (Windows, MacOS, and Linux) from the Eclipse foundation homepage: `www.eclipse.org/downloads/`

There are many variations available; the following is the recommended configuration for Android:

- Eclipse 3.4 or 3.5 (Galileo)
- Eclipse JDT plug-in
- WST

WST and the JDT plug-in are included in most Eclipse IDE packages.

FIGURE 4 (Meier Book, Page 19)

Using the Linux Portion of the Android Framework which includes built-in command line commands such as CAT, VI, CP, MV and RM, One can Create Arbitrary Objects, Manage Arbitrary Objects and Deploy these Arbitrary Objects.

Patent # 7,716,629 Claim Chart for the Android Framework

We will use the book “Professional Android 2 Application Development”, ISBN 978-0-470-5656552-0 written by Reto Meier, who described has “Reto Meier is a software developer who has been involved in Android since its initial release in 2007. He is an Android Developer Advocate at Google”, we will refer to this book from now on as “Meier Book”.

Claim 1:

“A system for generating a computer application on a host system in an arbitrary object framework that separates a content of said computer application, a form of said computer application, and a functionality of said computer application, said system including a computer comprising a processor and a memory operably coupled to said processor, said memory being configured for storing a computer program executable by said processor, said computer program comprising: “

“a first set of executable instructions for creating arbitrary objects with corresponding arbitrary names of content objects”

”used in generating said content of said computer application, “

“form objects used in defining said form of said computer application,”

“and function objects used in executing said functionality of said computer application each arbitrary object being separate from each other arbitrary object;”

“a second set of executable instructions for managing said arbitrary objects in an arbitrary object library; “

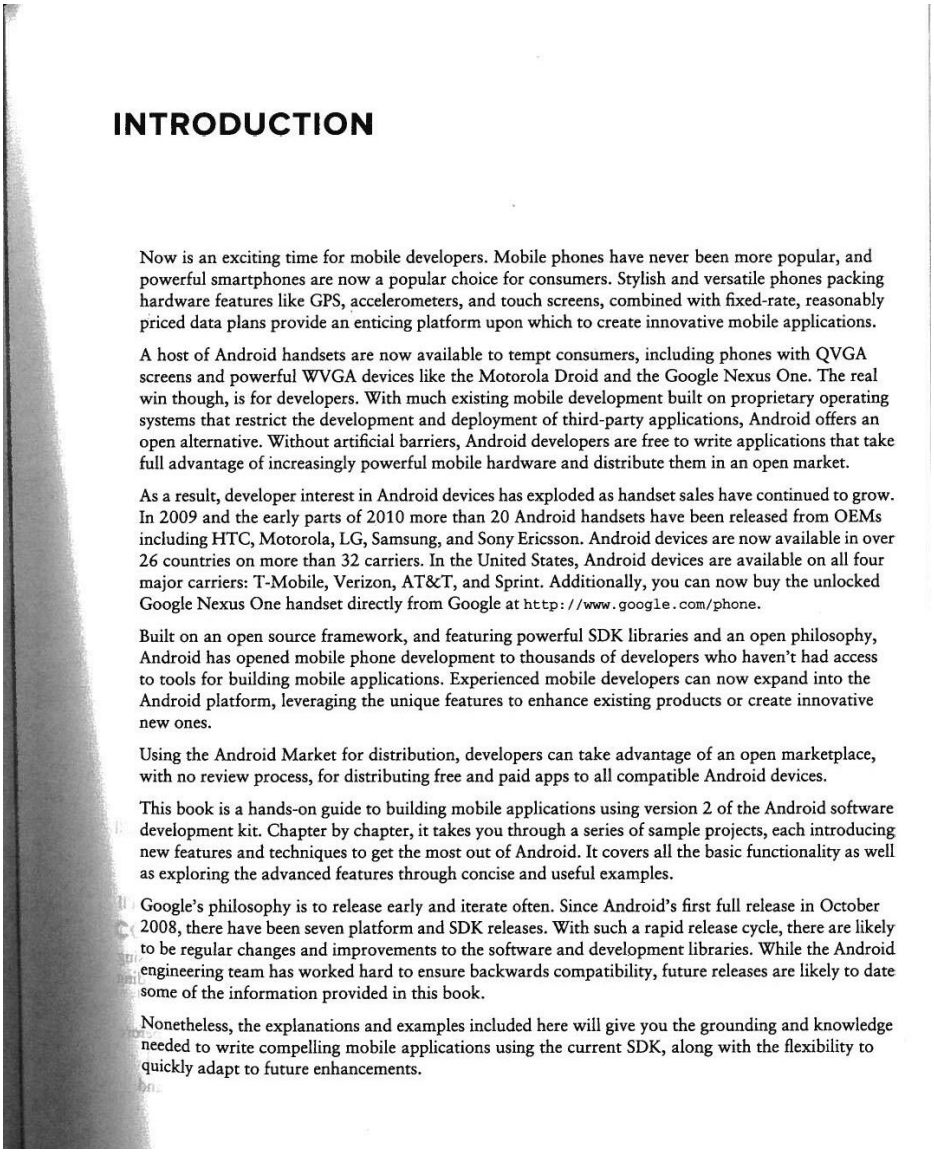
“and a third set of executable instructions for deploying said arbitrary objects from said arbitrary object library into a design framework to create said computer application.”

Claim 1 Preamble

"A system for generating a computer application on a host system in an arbitrary object framework that separates a content of said computer application, a form of said computer application, and a functionality of said computer application, said system including a computer comprising a processor and a memory operably coupled to said processor, said memory being configured for storing a computer program executable by said processor, said computer program comprising: "

-Android is a framework of arbitrary objects which separates form, function and content of a computer application, which runs on a host, either a smart phone , a tablet PC or even a virtual machine on a PC.

Meier Book, Page xvii (more specifically 4th paragraph) and Page 4 (more specifically 7th bullet) essentially describes what is known in the industry as a framework.



INTRODUCTION

Now is an exciting time for mobile developers. Mobile phones have never been more popular, and powerful smartphones are now a popular choice for consumers. Stylish and versatile phones packing hardware features like GPS, accelerometers, and touch screens, combined with fixed-rate, reasonably priced data plans provide an enticing platform upon which to create innovative mobile applications.

A host of Android handsets are now available to tempt consumers, including phones with QVGA screens and powerful WVGA devices like the Motorola Droid and the Google Nexus One. The real win though, is for developers. With much existing mobile development built on proprietary operating systems that restrict the development and deployment of third-party applications, Android offers an open alternative. Without artificial barriers, Android developers are free to write applications that take full advantage of increasingly powerful mobile hardware and distribute them in an open market.

As a result, developer interest in Android devices has exploded as handset sales have continued to grow. In 2009 and the early parts of 2010 more than 20 Android handsets have been released from OEMs including HTC, Motorola, LG, Samsung, and Sony Ericsson. Android devices are now available in over 26 countries on more than 32 carriers. In the United States, Android devices are available on all four major carriers: T-Mobile, Verizon, AT&T, and Sprint. Additionally, you can now buy the unlocked Google Nexus One handset directly from Google at `http://www.google.com/phone`.

Built on an open source framework, and featuring powerful SDK libraries and an open philosophy, Android has opened mobile phone development to thousands of developers who haven't had access to tools for building mobile applications. Experienced mobile developers can now expand into the Android platform, leveraging the unique features to enhance existing products or create innovative new ones.

Using the Android Market for distribution, developers can take advantage of an open marketplace, with no review process, for distributing free and paid apps to all compatible Android devices.

This book is a hands-on guide to building mobile applications using version 2 of the Android software development kit. Chapter by chapter, it takes you through a series of sample projects, each introducing new features and techniques to get the most out of Android. It covers all the basic functionality as well as exploring the advanced features through concise and useful examples.

Google's philosophy is to release early and iterate often. Since Android's first full release in October 2008, there have been seven platform and SDK releases. With such a rapid release cycle, there are likely to be regular changes and improvements to the software and development libraries. While the Android engineering team has worked hard to ensure backwards compatibility, future releases are likely to date some of the information provided in this book.

Nonetheless, the explanations and examples included here will give you the grounding and knowledge needed to write compelling mobile applications using the current SDK, along with the flexibility to quickly adapt to future enhancements.

FIGURE 5 (Meier Book, Page XVII)

stack produced and supported by the Open Handset Alliance and designed to operate on any handset that meets the requirements. Google has now released its first direct-to-consumer handset, the Nexus 1, but this device remains simply one hardware implementation running on the Android platform.

ANDROID: AN OPEN PLATFORM FOR MOBILE DEVELOPMENT

Google's Andy Rubin describes Android as:

> *The first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation.* (`http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html`)

Put simply, Android is a combination of three components:

- A free, open-source operating system for mobile devices
- An open-source development platform for creating mobile applications
- Devices, particularly mobile phones, that run the Android operating system and the applications created for it

More specifically, Android is made up of several necessary and dependent parts, including the following:

- A hardware reference design that describes the capabilities required for a mobile device to support the software stack.
- A Linux operating system kernel that provides low-level interface with the hardware, memory management, and process control, all optimized for mobile devices.
- Open-source libraries for application development, including SQLite, WebKit, OpenGL, and a media manager.
- A run time used to execute and host Android applications, including the Dalvik virtual machine and the core libraries that provide Android-specific functionality. The run time is designed to be small and efficient for use on mobile devices.
- An application framework that agnostically exposes system services to the application layer, including the window manager and location manager, content providers, telephony, and sensors.
- A user interface framework used to host and launch applications.
- Preinstalled applications shipped as part of the stack.
- A software development kit used to create applications, including tools, plug-ins, and documentation.

What really makes Android compelling is its open philosophy, which ensures that you can fix any deficiencies in user interface or native application design by writing an extension or replacement. Android

FIGURE 6 (Meier Book, Page 4)

Patent # 7,716,629 Claim Chart for the Android Framework
Meier Book, Page 28 explains how the UI (Form) "decouples" the application logic (function) from your UI design (form) on its first paragraph.

Listing 2-2 shows the UI layout defined in the `main.xml` file created by the Android project template.

LISTING 2-2: Hello World layout resource

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, HelloWorld"
  />
</LinearLayout>
```

Defining your UI in XML and inflating it is the preferred way of implementing your user interfaces, as it neatly decouples your application logic from your UI design.

To get access to your UI elements in code, you add identifier attributes to them in the XML definition. You can then use the `findViewById` method to return a reference to each named item. The following XML snippet shows an ID attribute added to the Text View widget in the Hello World template:

```
<TextView
  android:id="@+id/myTextView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Hello World, HelloWorld"
/>
```

And the following snippet shows how to get access to it in code:

```
TextView myTextView = (TextView)findViewById(R.id.myTextView);
```

Alternatively (although it's not generally considered good practice), you can create your layout directly in code, as shown in Listing 2-3.

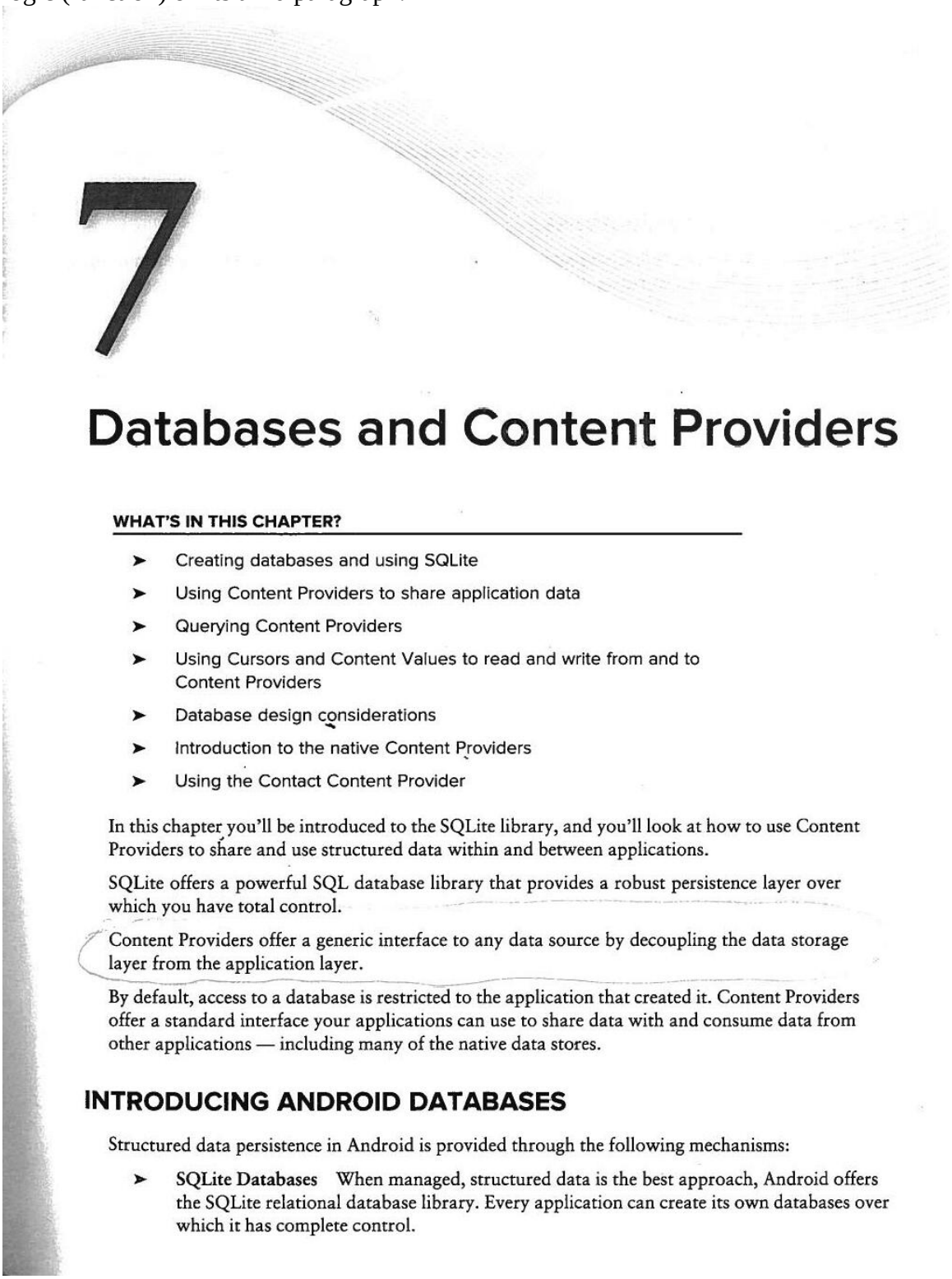
LISTING 2-3: Creating layouts in code

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  LinearLayout.LayoutParams lp;
  lp = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
                                     LayoutParams.FILL_PARENT);

  LinearLayout.LayoutParams textViewLP;
  textViewLP = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
                                             LayoutParams.WRAP_CONTENT);

  LinearLayout ll = new LinearLayout(this);
  ll.setOrientation(LinearLayout.VERTICAL);

  TextView myTextView = new TextView(this);
  myTextView.setText("Hello World, HelloWorld");
```

FIGURE 7 (Meier Book, Page 28)

Patent # 7,716,629 Claim Chart for the Android Framework

Meier Book, Page 209 defines how the "Content Providers" "decouples" data storage (content) from your application logic (function) on its third paragraph.



7

Databases and Content Providers

WHAT'S IN THIS CHAPTER?

- Creating databases and using SQLite
- Using Content Providers to share application data
- Querying Content Providers
- Using Cursors and Content Values to read and write from and to Content Providers
- Database design considerations
- Introduction to the native Content Providers
- Using the Contact Content Provider

In this chapter you'll be introduced to the SQLite library, and you'll look at how to use Content Providers to share and use structured data within and between applications.

SQLite offers a powerful SQL database library that provides a robust persistence layer over which you have total control.

Content Providers offer a generic interface to any data source by decoupling the data storage layer from the application layer.

By default, access to a database is restricted to the application that created it. Content Providers offer a standard interface your applications can use to share data with and consume data from other applications — including many of the native data stores.

INTRODUCING ANDROID DATABASES

Structured data persistence in Android is provided through the following mechanisms:

- **SQLite Databases** When managed, structured data is the best approach, Android offers the SQLite relational database library. Every application can create its own databases over which it has complete control.

FIGURE 8 (Meier Book, Page 209)

Patent # 7,716,629 Claim Chart for the Android Framework
Android's objects can be arbitrary objects. We created a sample application using the Android Software Developers' Kit (Android SDK). In the following example:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/textview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:text="@string/hello"/>
```

The Arbitrary Object "textview" is being called by its name (android:id="@+id/textview"), and additionally has three parameters.

When compiling and running the application, the compilation goes fine, and the application runs fine (it prints "Alo mundo,Hello Android!) on the Android device as seen in here"

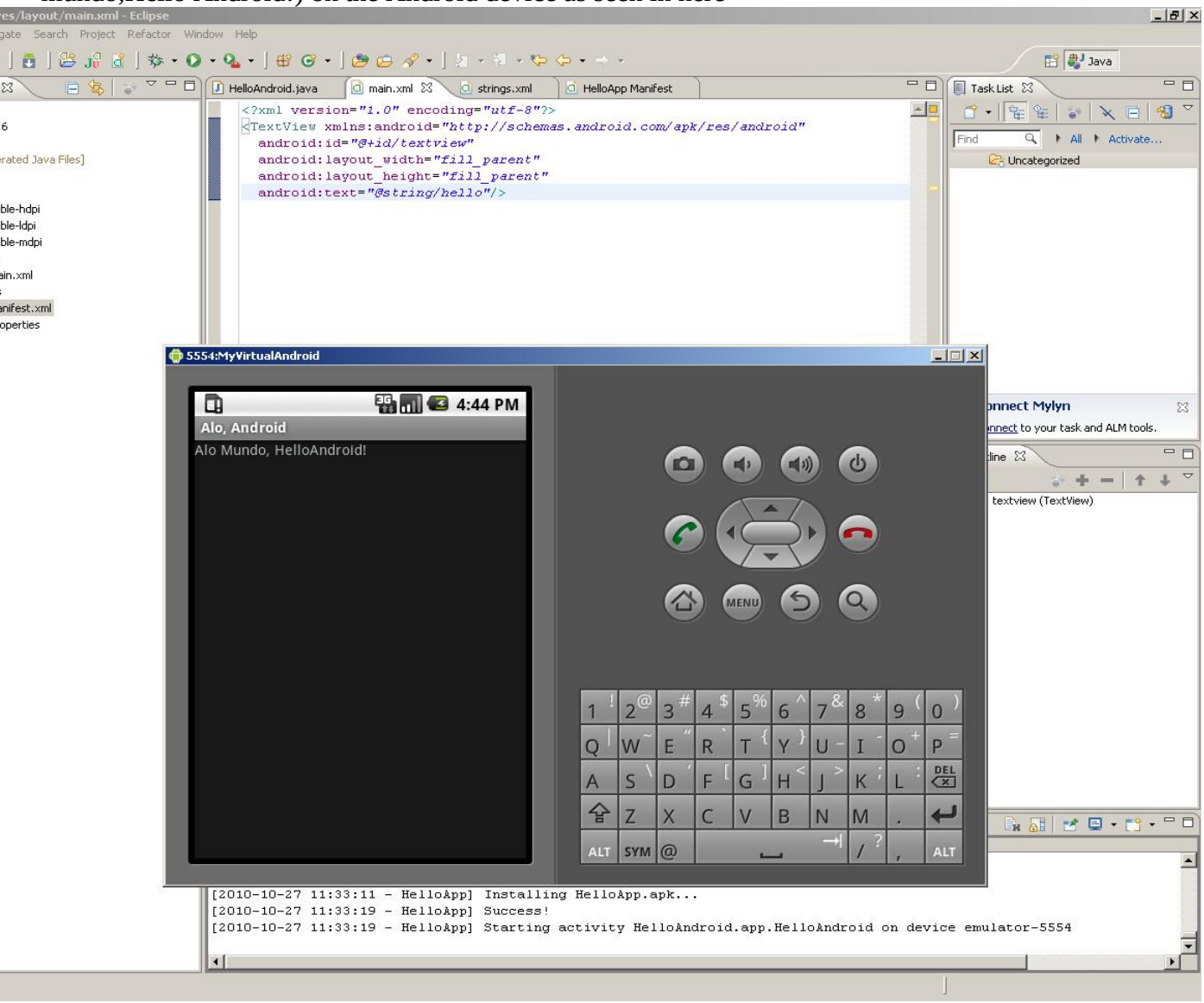


FIGURE 9 (Arbitrary Objects called with parameters)

However, when we modify the call to “textview” to include just the name of the Arbitrary Object, the call will look this way:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/textview"
  />
```

When making the above modification, the program compiles fine and the application runs normally, as can be seen on this screen print (We will refer to this later as “Sample Application”)

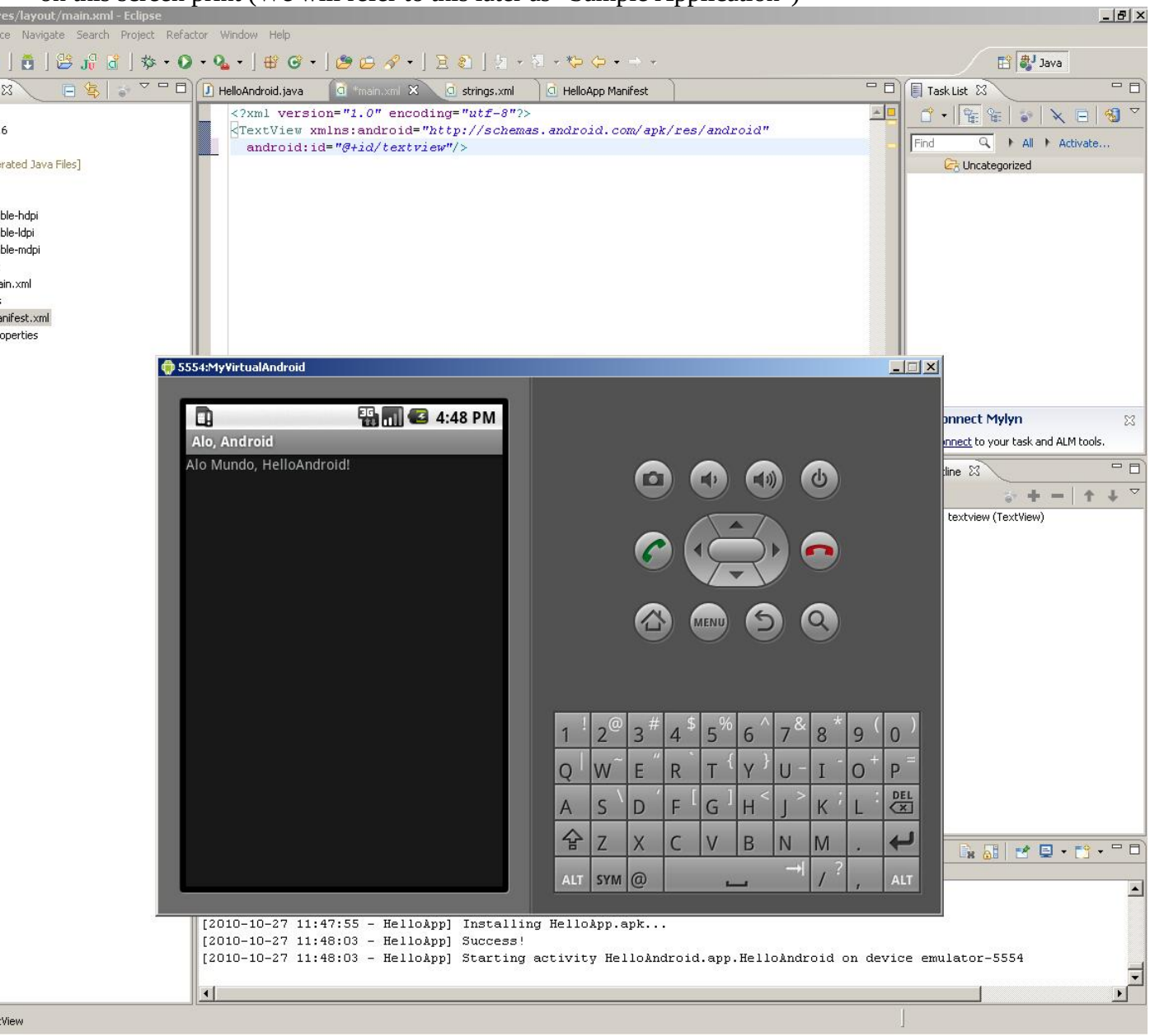


FIGURE 10 (Arbitrary Objects called by name only)

Claim 1 Element 1

“a first set of executable instructions for creating arbitrary objects with corresponding arbitrary names of content objects”

The creation of those Arbitrary objects can be using command line tools, such as a text editor, or using the recommended tool , which is named Eclipse. Using a text editor is a lot more work because all of the files needed to create the object need to be edited correctly and placed on the correct location (folders), in order for the compiler to work. Using Eclipse facilitates the process but it is not necessary to create the Arbitrary Objects.

The steps for generating a Computer application on the Android Framework when not using Eclipse are:

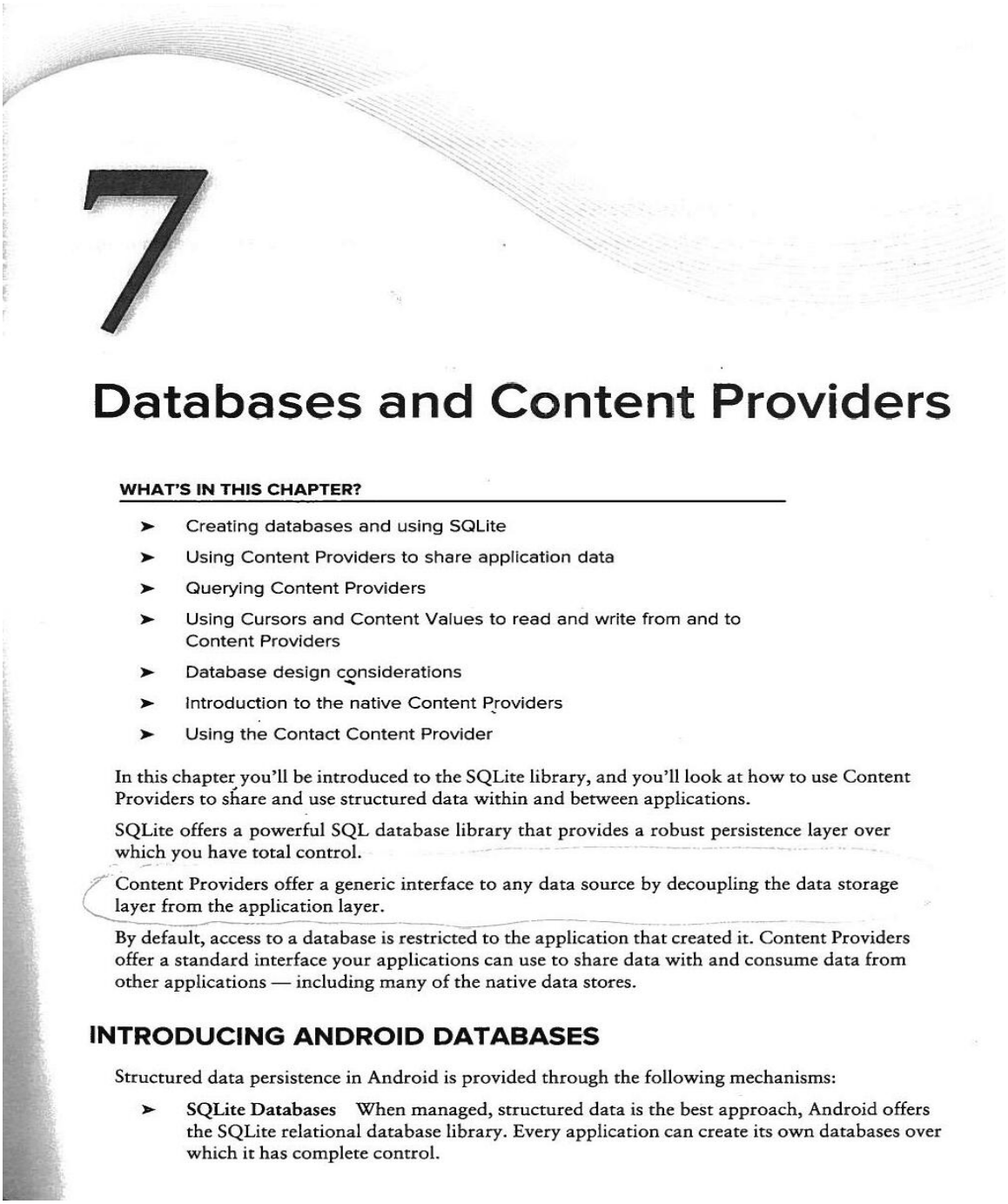
-Creation of the application either by manually creating each source file with a text editor
-Compiling of the application using a Java Compiler
-Creating a “package” (A file which’s extension is .APK”, using the Android SDK named apkbuilder.bat , which is distributed with the Android SDK.
-Moving that package file to the Android device, either by using Google’s “Android Market” application (in the case of a SmartPhone), or using the buil tin android application named “File Manager”, clicking on the file, and Android will then install it on the device.

When using eclipse, all is necessary is to click on the “Run” menu – that is what was done on the application described earlier.

Claim 1 Element 2

" used in generating said content of said computer application, "

The content can be generated by the "Content Provider" feature of Android. Meier Book, Pages 209-211 is a description of how it works. The title of the chapter, and throughout the whole text, it clarifies that the Android Framework can be used for generating the content:



7

Databases and Content Providers

WHAT'S IN THIS CHAPTER?

- Creating databases and using SQLite
- Using Content Providers to share application data
- Querying Content Providers
- Using Cursors and Content Values to read and write from and to Content Providers
- Database design considerations
- Introduction to the native Content Providers
- Using the Contact Content Provider

In this chapter you'll be introduced to the SQLite library, and you'll look at how to use Content Providers to share and use structured data within and between applications.

SQLite offers a powerful SQL database library that provides a robust persistence layer over which you have total control.

Content Providers offer a generic interface to any data source by decoupling the data storage layer from the application layer.

By default, access to a database is restricted to the application that created it. Content Providers offer a standard interface your applications can use to share data with and consume data from other applications — including many of the native data stores.

INTRODUCING ANDROID DATABASES

Structured data persistence in Android is provided through the following mechanisms:

- **SQLite Databases** When managed, structured data is the best approach, Android offers the SQLite relational database library. Every application can create its own databases over which it has complete control.

FIGURE 11 (Meier Book, Page 209)

- **Content Providers** Content Providers offer a generic, well-defined interface for using and sharing data.

Introducing SQLite Databases

Using SQLite you can create independent relational databases for your applications. Use them to store and manage complex, structured application data.

Android databases are stored in the `/data/data/<package_name>/databases` folder on your device (or emulator). By default all databases are private, accessible only by the application that created them.

Database design is a big topic that deserves more thorough coverage than is possible within this book. It is worth highlighting that standard database best practices still apply in Android. In particular, when you're creating databases for resource-constrained devices (such as mobile phones), it's important to normalize your data to reduce redundancy.

Introducing Content Providers

Content Providers provide an interface for publishing and consuming data, based around a simple URI addressing model using the `content://` schema. They let you decouple the application layer from the data layer, making your applications data-source agnostic by hiding the underlying data source.

Shared Content Providers can be queried for results, existing records updated or deleted, and new records added. Any application with the appropriate permissions can add, remove, or update data from any other application — including from the native Android databases.

Many native databases are available as Content Providers, accessible by third-party applications, including the phone's contact manager, media store, and other native databases as described later in this chapter.

By publishing your own data sources as Content Providers, you make it possible for you (and other developers) to incorporate and extend your data in new applications.

INTRODUCING SQLite

SQLite is a well regarded relational database management system (RDBMS). It is:

- Open-source
- Standards-compliant
- Lightweight
- Single-tier

It has been implemented as a compact C library that's included as part of the Android software stack.

By being implemented as a library, rather than running as a separate ongoing process, each SQLite database is an integrated part of the application that created it. This reduces external dependencies, minimizes latency, and simplifies transaction locking and synchronization.

SQLite has a reputation for being extremely reliable and is the database system of choice for many consumer electronic devices, including several MP3 players, the iPhone, and the iPod Touch.

FIGURE 12 (Meier Book, Page 210)

Lightweight and powerful, SQLite differs from many conventional database engines by loosely typing each column, meaning that column values are not required to conform to a single type. Instead, each value is typed individually for each row. As a result, type checking isn't necessary when assigning or extracting values from each column within a row.

For more comprehensive coverage of SQLite, including its particular strengths and limitations, check out the official site at `http://www.sqlite.org/`

CURSORS AND CONTENT VALUES

`ContentValues` are used to insert new rows into tables. Each Content Values object represents a single table row as a map of column names to values.

Queries in Android are returned as `Cursor` objects. Rather than extracting and returning a copy of the result values, Cursors are pointers to the result set within the underlying data. Cursors provide a managed way of controlling your position (row) in the result set of a database query.

The Cursor class includes a number of navigation functions including, but not limited to, the following:

- `moveToFirst` Moves the cursor to the first row in the query result
- `moveToNext` Moves the cursor to the next row
- `moveToPrevious` Moves the cursor to the previous row
- `getCount` Returns the number of rows in the result set
- `getColumnIndexOrThrow` Returns the index for the column with the specified name (throwing an exception if no column exists with that name)
- `getColumnName` Returns the name of the specified column index
- `getColumnNames` Returns a string array of all the column names in the current Cursor
- `moveToPosition` Moves the Cursor to the specified row
- `getPosition` Returns the current Cursor position

Android provides a convenient mechanism for simplifying the management of Cursors within your Activities. The `startManagingCursor` method integrates the Cursor's lifetime into the calling Activity's. When you've finished with the Cursor, call `stopManagingCursor` to do just that.

Later in this chapter you'll learn how to query a database and how to extract specific row/column values from the resulting Cursors.

WORKING WITH SQLite DATABASES

It's good practice to create a helper class to simplify your database interactions.

The following section shows you how to create a database adapter class for your database. This abstraction layer encapsulates your database interactions. It will provide intuitive, strongly typed methods for adding, removing, and updating items. A database adapter should also handle queries and expose methods for creating, opening, and closing the database.

FIGURE 13 (Meier Book, Page 211)

Claim 1 Element 3

”used in generating said content of said computer application, “

Form in the Android Framework normally is defined on the Resource folder, referred to as “R” or “Layouts”, having the default “layout.xml” as the Form, but this name can be changed. On Meier Book, Page 63 and 64 describes how they work. More specifically , the last three paragraphs of Page 63 define Layouts as your “Presentation Layer” and “User Interface”, which is Form.


```
        <item name="attributeName">value</item>
    </style>
</resources>
```

Styles support inheritance using the `parent` attribute on the `<style>` tag, making it easy to create simple variations.

The following example shows two styles that can also be used as a theme: a base style that sets several text properties and a second style that modifies the first to specify a smaller font.

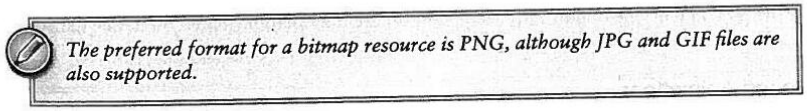
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="BaseText">
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#111</item>
  </style>
  <style name="SmallText" parent="BaseText">
    <item name="android:textSize">8sp</item>
  </style>
</resources>
```

Drawables

Drawable resources include bitmaps and NinePatch (stretchable PNG) images. They also include complex composite Drawables, such as `LevelListDrawables` and `StateListDrawables` that can be defined in XML.

Both NinePatch Drawables and complex composite resources are covered in more detail in the next chapter.

All Drawables are stored as individual files in the `res/drawable` folder. The resource identifier for a Drawable resource is the lowercase file name without an extension.



The preferred format for a bitmap resource is PNG, although JPG and GIF files are also supported.

Layouts

Layout resources let you decouple your presentation layer by designing user interface layouts in XML rather than constructing them in code.

The most common use of a layout is for defining the user interface for an Activity. Once defined in XML, the layout is “inflated” within an Activity using `setContentView`, usually within the `onCreate` method. You can also reference layouts from within other layout resources, such as layouts for each row in a List View. More detailed information on using and creating layouts in Activities can be found in Chapter 4.

Using layouts to create your screens is best-practice UI design in Android. The decoupling of the layout from the code lets you create optimized layouts for different hardware configurations, such as varying screen sizes, orientation, or the presence of keyboards and touchscreens.

FIGURE 14 (Meier Book, Page 63)

Each layout definition is stored in a separate file, each containing a single layout, in the `res/layout` folder. The file name then becomes the resource identifier.

A thorough explanation of layout containers and View elements is included in the next chapter, but as an example Listing 3-2 shows the layout created by the New Project Wizard. It uses a Linear Layout (described in more detail in Chapter 4) as a layout container for a Text View that displays the "Hello World" greeting.

LISTING 3-2: Hello World layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World!"
  />
</LinearLayout>
```

Animations

Android supports two types of animation. *Tweened animations* can be used to rotate, move, stretch, and fade a View; or you can create frame-by-frame animations to display a sequence of Drawable images. A comprehensive overview of creating, using, and applying animations can be found in Chapter 15.

Defining animations as external resources enables you to reuse the same sequence in multiple places and provides you with the opportunity to present different animations based on device hardware or orientation.

Tweened Animations

Each tweened animation is stored in a separate XML file in the project's `res/anim` folder. As with layouts and Drawable resources, the animation's file name is used as its resource identifier.

An animation can be defined for changes in `alpha` (fading), `scale` (scaling), `translate` (movement), or `rotate` (rotation).

Table 3-1 shows the valid attributes, and attribute values, supported by each animation type.

You can create a combination of animations using the `set` tag. An animation set contains one or more animation transformations and supports various additional tags and attributes to customize when and how each animation within the set is run.

The following list shows some of the set tags available.

- `duration` Duration of the animation in milliseconds.
- `startOffset` Millisecond delay before the animation starts.

FIGURE 15 (Meier Book, Page 64)

Patent # 7,716,629 Claim Chart for the Android Framework
Also on Meier Book, Page 50, it describes “Activities” as Presentation Layer (which can be Form) of Android Applications.

Mobile devices come in a large variety of shapes and sizes and are used across the world. In this chapter you'll learn how to externalize resources to ensure your applications run seamlessly on different hardware (particularly different screen resolutions and pixel densities), in different countries, and supporting multiple languages.

Next you'll examine the Application class, and learn how to extend it to provide a place for storing application state values.

Arguably the most important of the Android building blocks, the `Activity` class forms the basis for all your user interface screens. You'll learn how to create new Activities and gain an understanding of their life cycles and how they affect the application lifetime.

Finally, you'll be introduced to some of the Activity subclasses that simplify resource management for some common user interface components such as maps and lists.

WHAT MAKES AN ANDROID APPLICATION?

Android applications consist of loosely coupled components, bound by an application manifest that describes each component and how they all interact, as well as the application metadata including its hardware and platform requirements.

The following six components provide the building blocks for your applications:

- **Activities** Your application's presentation layer. Every screen in your application will be an extension of the `Activity` class. Activities use Views to form graphical user interfaces that display information and respond to user actions. In terms of desktop development, an Activity is equivalent to a Form. You'll learn more about Activities later in this chapter.
- **Services** The invisible workers of your application. Service components run in the background, updating your data sources and visible Activities and triggering Notifications. They're used to perform regular processing that needs to continue even when your application's Activities aren't active or visible. You'll learn how to create Services in Chapter 9.
- **Content Providers** Shareable data stores. Content Providers are used to manage and share application databases. They're the preferred means of sharing data across application boundaries. This means that you can configure your own Content Providers to permit access from other applications and use Content Providers exposed by others to access their stored data. Android devices include several native Content Providers that expose useful databases like the media store and contact details. You'll learn how to create and use Content Providers in Chapter 7.
- **Intents** An inter-application message-passing framework. Using Intents you can broadcast messages system-wide or to a target Activity or Service, stating your intention to have an action performed. The system will then determine the target(s) that will perform any actions as appropriate.
- **Broadcast Receivers** Intent broadcast consumers. If you create and register a Broadcast Receiver, your application can listen for broadcast Intents that match specific filter

FIGURE 16 (Meier Book, Page 50)

Claim 1 Element 4

"and function objects used in executing said functionality of said computer application each arbitrary object being separate from each other arbitrary object;"

Meier Book, Page 50, second bullet, describes "Services" as "The invisible workers of the application", (Which can be Functionality):

Mobile devices come in a large variety of shapes and sizes and are used across the world. In this chapter you'll learn how to externalize resources to ensure your applications run seamlessly on different hardware (particularly different screen resolutions and pixel densities), in different countries, and supporting multiple languages.

Next you'll examine the Application class, and learn how to extend it to provide a place for storing application state values.

Arguably the most important of the Android building blocks, the `Activity` class forms the basis for all your user interface screens. You'll learn how to create new Activities and gain an understanding of their life cycles and how they affect the application lifetime.

Finally, you'll be introduced to some of the Activity subclasses that simplify resource management for some common user interface components such as maps and lists.

WHAT MAKES AN ANDROID APPLICATION?

Android applications consist of loosely coupled components, bound by an application manifest that describes each component and how they all interact, as well as the application metadata including its hardware and platform requirements.

The following six components provide the building blocks for your applications:

- **Activities** Your application's presentation layer. Every screen in your application will be an extension of the `Activity` class. Activities use Views to form graphical user interfaces that display information and respond to user actions. In terms of desktop development, an Activity is equivalent to a Form. You'll learn more about Activities later in this chapter.
- **Services** The invisible workers of your application. Service components run in the background, updating your data sources and visible Activities and triggering Notifications. They're used to perform regular processing that needs to continue even when your application's Activities aren't active or visible. You'll learn how to create Services in Chapter 9.
- **Content Providers** Shareable data stores. Content Providers are used to manage and share application databases. They're the preferred means of sharing data across application boundaries. This means that you can configure your own Content Providers to permit access from other applications and use Content Providers exposed by others to access their stored data. Android devices include several native Content Providers that expose useful databases like the media store and contact details. You'll learn how to create and use Content Providers in Chapter 7.
- **Intents** An inter-application message-passing framework. Using Intents you can broadcast messages system-wide or to a target Activity or Service, stating your intention to have an action performed. The system will then determine the target(s) that will perform any actions as appropriate.
- **Broadcast Receivers** Intent broadcast consumers. If you create and register a Broadcast Receiver, your application can listen for broadcast Intents that match specific filter

FIGURE 17 (Meier Book, Page 50)

Claim 1 Element 5

“a second set of executable instructions for managing said arbitrary objects in an arbitrary object library; “

The object library, when using eclipse is described as the “Workspace” of the Android Applications. If not using Eclipse, the management of these objects can be done manually by using normal command-line commands , such as “Copy”, “Rename” , “Delete” or “Move”, inside of a folder structure which is mandated by the Android SDK so the Computer Application can be built properly.

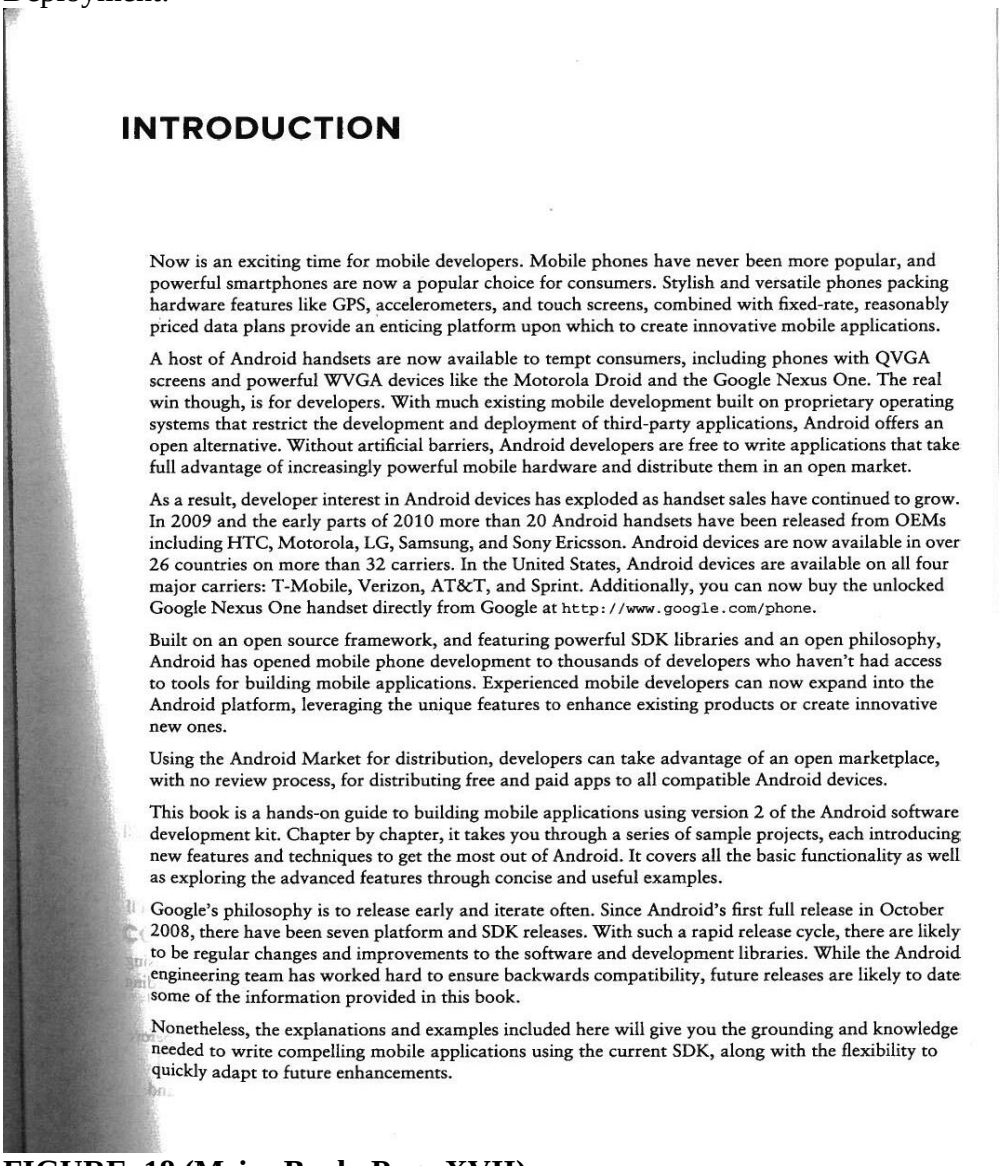
Claim 1 Element 6

"and a third set of executable instructions for deploying said arbitrary objects from said arbitrary object library into a design framework to create said computer application."

These Arbitrary objects are deployed by building the complete application as described earlier :

-Moving that package file to the Android device, either by using Google's "Android Market" application (in the case of a SmartPhone), or using the buil tin android application named "File Manager", clicking on the file, and Android will then install it on the device.

The Meier Book, page XVII, on its 5th Paragraph, describes that the Android Market can be used for distribution, or Deployment:



INTRODUCTION

Now is an exciting time for mobile developers. Mobile phones have never been more popular, and powerful smartphones are now a popular choice for consumers. Stylish and versatile phones packing hardware features like GPS, accelerometers, and touch screens, combined with fixed-rate, reasonably priced data plans provide an enticing platform upon which to create innovative mobile applications.

A host of Android handsets are now available to tempt consumers, including phones with QVGA screens and powerful WVGA devices like the Motorola Droid and the Google Nexus One. The real win though, is for developers. With much existing mobile development built on proprietary operating systems that restrict the development and deployment of third-party applications, Android offers an open alternative. Without artificial barriers, Android developers are free to write applications that take full advantage of increasingly powerful mobile hardware and distribute them in an open market.

As a result, developer interest in Android devices has exploded as handset sales have continued to grow. In 2009 and the early parts of 2010 more than 20 Android handsets have been released from OEMs including HTC, Motorola, LG, Samsung, and Sony Ericsson. Android devices are now available in over 26 countries on more than 32 carriers. In the United States, Android devices are available on all four major carriers: T-Mobile, Verizon, AT&T, and Sprint. Additionally, you can now buy the unlocked Google Nexus One handset directly from Google at `http://www.google.com/phone`.

Built on an open source framework, and featuring powerful SDK libraries and an open philosophy, Android has opened mobile phone development to thousands of developers who haven't had access to tools for building mobile applications. Experienced mobile developers can now expand into the Android platform, leveraging the unique features to enhance existing products or create innovative new ones.

Using the Android Market for distribution, developers can take advantage of an open marketplace, with no review process, for distributing free and paid apps to all compatible Android devices.

This book is a hands-on guide to building mobile applications using version 2 of the Android software development kit. Chapter by chapter, it takes you through a series of sample projects, each introducing new features and techniques to get the most out of Android. It covers all the basic functionality as well as exploring the advanced features through concise and useful examples.

Google's philosophy is to release early and iterate often. Since Android's first full release in October 2008, there have been seven platform and SDK releases. With such a rapid release cycle, there are likely to be regular changes and improvements to the software and development libraries. While the Android engineering team has worked hard to ensure backwards compatibility, future releases are likely to date some of the information provided in this book.

Nonetheless, the explanations and examples included here will give you the grounding and knowledge needed to write compelling mobile applications using the current SDK, along with the flexibility to quickly adapt to future enhancements.

FIGURE 18 (Meier Book, Page XVII)

Claim 4

“The system of claim 1, wherein the third set of executable instructions are for deploying arbitrary objects locally.”

The Sample Application demonstrated on Claim 1, Element 1, makes use of arbitrary objects locally.

Claim 8

“The system of claim 1, wherein the third set of executable instructions include instructions to access and deploy arbitrary objects into said design framework using said corresponding arbitrary names”

The Android Framework enables access and deployment of said arbitrary names into said design framework.

Patent # 7,716,629 Claim Chart for the Android Framework

Claim 10

"The system of claim 1, further comprising executable instructions for caching objects."

The Android Framework caches objects. In this picture showing the Android terminal emulation application, the response to the command "ls" shows a subdirectory named "cache":

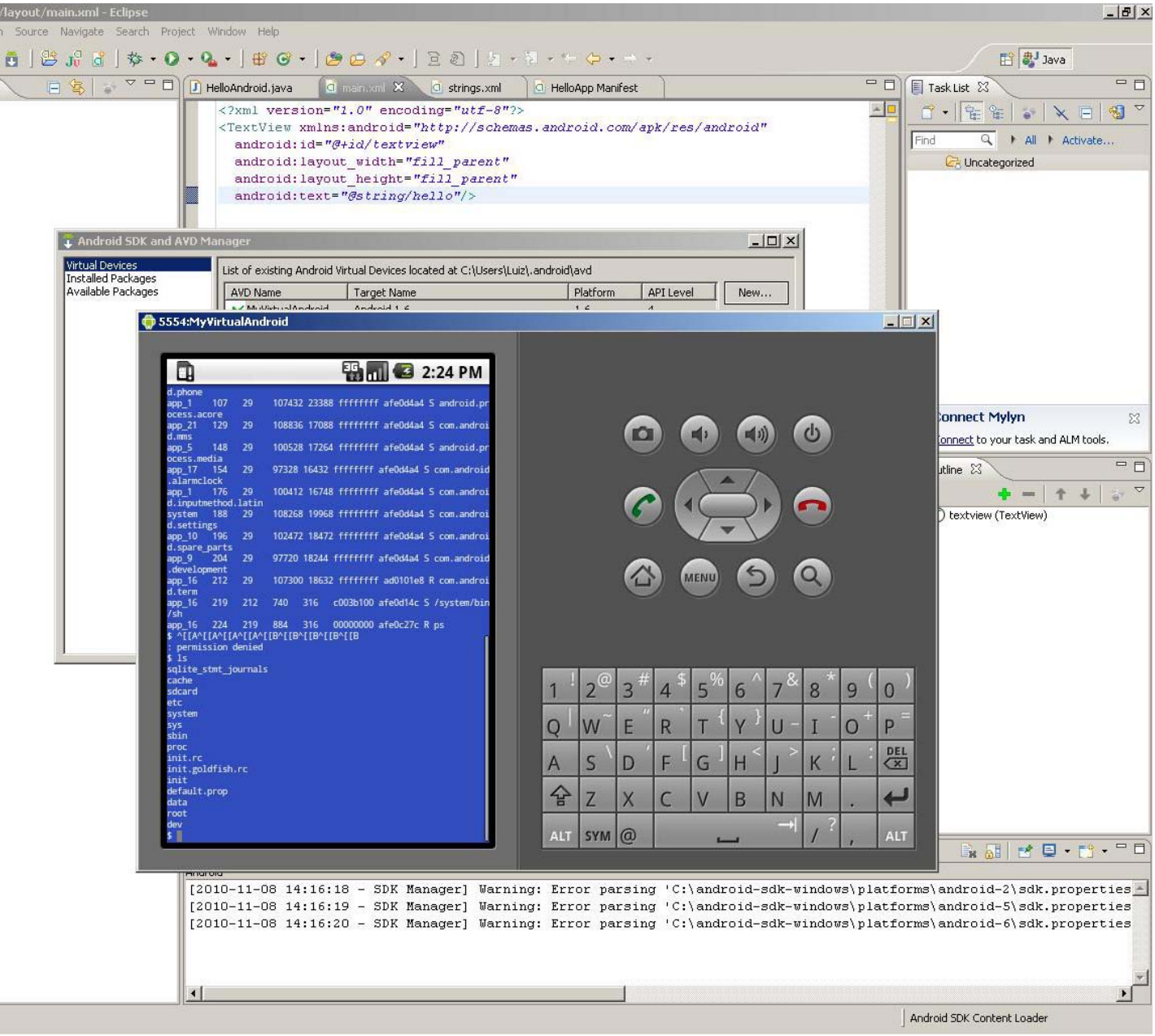


FIGURE 19 (use cache)

Claim 12

“The system of claim 1, further comprising executable instructions for generating arbitrary objects in a programming language that is compatible and supported by said host system”

The Android Framework allows such arbitrary objects to be developed in Java, which is a programming language supported by the host system.

Claim 21

“A system for generating a computer application on a host system in an arbitrary object framework that separates a content of said computer application, a form of said computer application, and a functionality of said computer application, said system including a computer comprising a processor and a memory operably coupled to said processor, said memory being configured for storing a computer program executable by said processor, said computer program comprising:”

“a first set of executable instructions for creating arbitrary objects with corresponding arbitrary names”

“of content objects used in generating said content of said computer application,”

”form objects used in defining said form of said computer application,“

“and function objects used in executing said functionality of said computer application,”

“each arbitrary object being callable by name only,”

”each arbitrary object being independently modifiable without corresponding modifications being made to any other arbitrary object, “

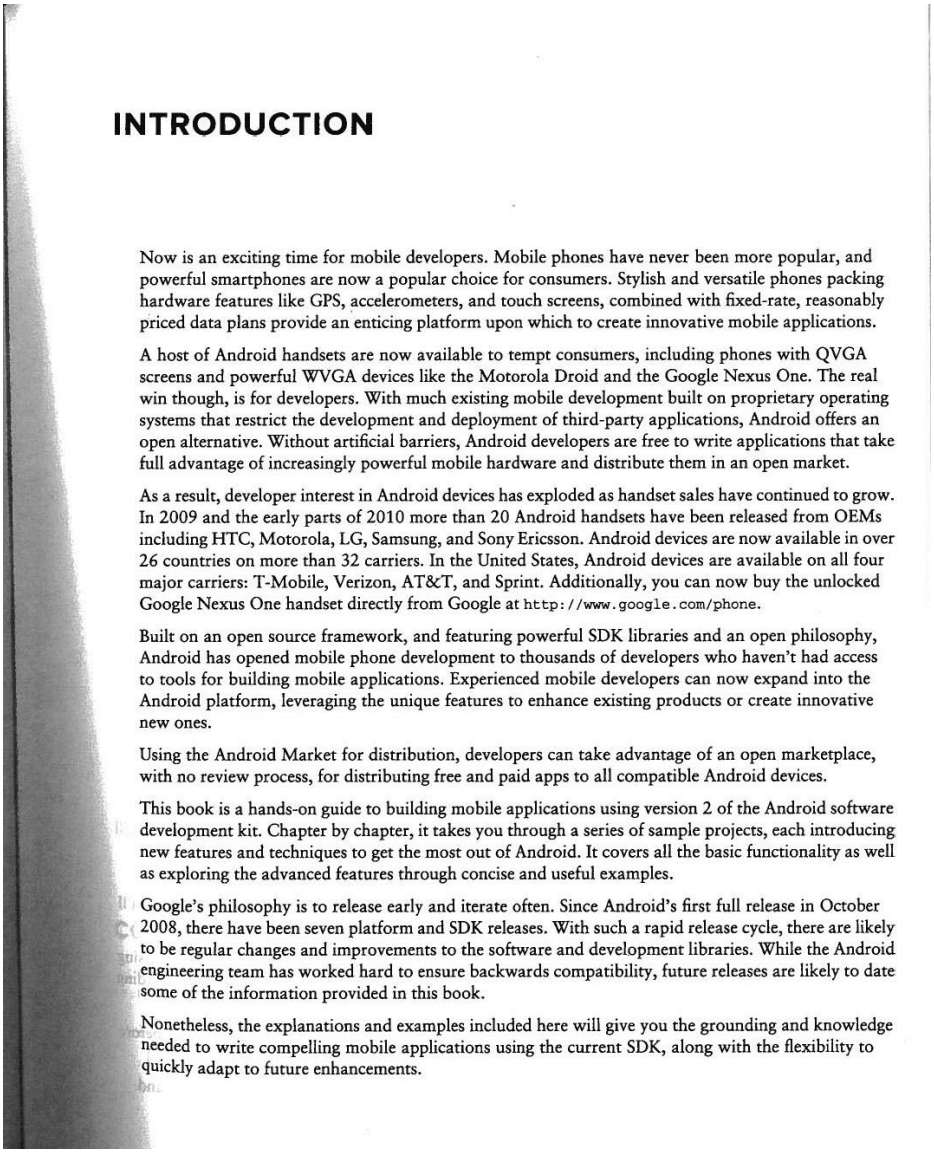
“and each arbitrary object further being interchangeable with other arbitrary objects;”

Claim 21 Preamble

"A system for generating a computer application on a host system in an arbitrary object framework that separates a content of said computer application, a form of said computer application, and a functionality of said computer application, said system including a computer comprising a processor and a memory operably coupled to said processor, said memory being configured for storing a computer program executable by said processor, said computer program comprising:"

-Android is a framework of arbitrary objects which separates form, function and content of a computer application, which runs on a host, either a smart phone , a tablet PC or even a virtual machine on a PC.

Meier Book, Page xvii (more specifically 4th paragraph) and Page 4 (more specifically 7th bullet) essentially describes what is known in the industry as a framework.



INTRODUCTION

Now is an exciting time for mobile developers. Mobile phones have never been more popular, and powerful smartphones are now a popular choice for consumers. Stylish and versatile phones packing hardware features like GPS, accelerometers, and touch screens, combined with fixed-rate, reasonably priced data plans provide an enticing platform upon which to create innovative mobile applications.

A host of Android handsets are now available to tempt consumers, including phones with QVGA screens and powerful WVGA devices like the Motorola Droid and the Google Nexus One. The real win though, is for developers. With much existing mobile development built on proprietary operating systems that restrict the development and deployment of third-party applications, Android offers an open alternative. Without artificial barriers, Android developers are free to write applications that take full advantage of increasingly powerful mobile hardware and distribute them in an open market.

As a result, developer interest in Android devices has exploded as handset sales have continued to grow. In 2009 and the early parts of 2010 more than 20 Android handsets have been released from OEMs including HTC, Motorola, LG, Samsung, and Sony Ericsson. Android devices are now available in over 26 countries on more than 32 carriers. In the United States, Android devices are available on all four major carriers: T-Mobile, Verizon, AT&T, and Sprint. Additionally, you can now buy the unlocked Google Nexus One handset directly from Google at `http://www.google.com/phone`.

Built on an open source framework, and featuring powerful SDK libraries and an open philosophy, Android has opened mobile phone development to thousands of developers who haven't had access to tools for building mobile applications. Experienced mobile developers can now expand into the Android platform, leveraging the unique features to enhance existing products or create innovative new ones.

Using the Android Market for distribution, developers can take advantage of an open marketplace, with no review process, for distributing free and paid apps to all compatible Android devices.

This book is a hands-on guide to building mobile applications using version 2 of the Android software development kit. Chapter by chapter, it takes you through a series of sample projects, each introducing new features and techniques to get the most out of Android. It covers all the basic functionality as well as exploring the advanced features through concise and useful examples.

Google's philosophy is to release early and iterate often. Since Android's first full release in October 2008, there have been seven platform and SDK releases. With such a rapid release cycle, there are likely to be regular changes and improvements to the software and development libraries. While the Android engineering team has worked hard to ensure backwards compatibility, future releases are likely to date some of the information provided in this book.

Nonetheless, the explanations and examples included here will give you the grounding and knowledge needed to write compelling mobile applications using the current SDK, along with the flexibility to quickly adapt to future enhancements.

FIGURE 20 (Meier Book, Page XVII)

stack produced and supported by the Open Handset Alliance and designed to operate on any handset that meets the requirements. Google has now released its first direct-to-consumer handset, the Nexus 1, but this device remains simply one hardware implementation running on the Android platform.

ANDROID: AN OPEN PLATFORM FOR MOBILE DEVELOPMENT

Google's Andy Rubin describes Android as:

> *The first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation.* (`http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html`)

Put simply, Android is a combination of three components:

- A free, open-source operating system for mobile devices
- An open-source development platform for creating mobile applications
- Devices, particularly mobile phones, that run the Android operating system and the applications created for it

More specifically, Android is made up of several necessary and dependent parts, including the following:

- A hardware reference design that describes the capabilities required for a mobile device to support the software stack.
- A Linux operating system kernel that provides low-level interface with the hardware, memory management, and process control, all optimized for mobile devices.
- Open-source libraries for application development, including SQLite, WebKit, OpenGL, and a media manager.
- A run time used to execute and host Android applications, including the Dalvik virtual machine and the core libraries that provide Android-specific functionality. The run time is designed to be small and efficient for use on mobile devices.
- An application framework that agnostically exposes system services to the application layer, including the window manager and location manager, content providers, telephony, and sensors.
- A user interface framework used to host and launch applications.
- Preinstalled applications shipped as part of the stack.
- A software development kit used to create applications, including tools, plug-ins, and documentation.

What really makes Android compelling is its open philosophy, which ensures that you can fix any deficiencies in user interface or native application design by writing an extension or replacement. Android

FIGURE 21 (Meier Book, Page 4)

Patent # 7,716,629 Claim Chart for the Android Framework
Meier Book, Page 28 explains how the UI (Form) "decouples" the application logic (function) from your UI design (form) on its first paragraph.

Listing 2-2 shows the UI layout defined in the `main.xml` file created by the Android project template.

LISTING 2-2: Hello World layout resource

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, HelloWorld"
  />
</LinearLayout>
```

Defining your UI in XML and inflating it is the preferred way of implementing your user interfaces, as it neatly decouples your application logic from your UI design.

To get access to your UI elements in code, you add identifier attributes to them in the XML definition. You can then use the `findViewById` method to return a reference to each named item. The following XML snippet shows an ID attribute added to the Text View widget in the Hello World template:

```
<TextView
  android:id="@+id/myTextView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Hello World, HelloWorld"
/>
```

And the following snippet shows how to get access to it in code:

```
TextView myTextView = (TextView)findViewById(R.id.myTextView);
```

Alternatively (although it's not generally considered good practice), you can create your layout directly in code, as shown in Listing 2-3.

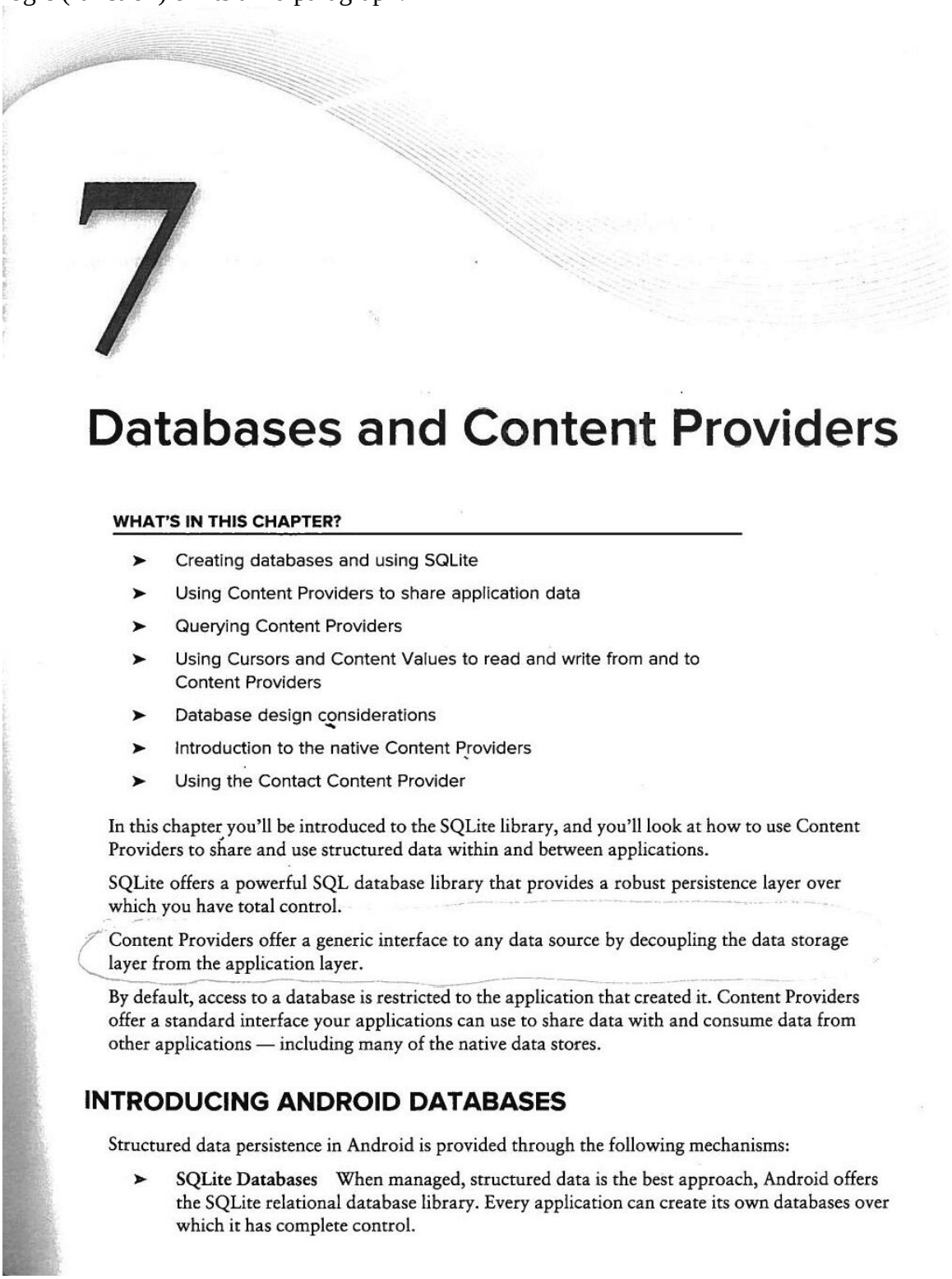
LISTING 2-3: Creating layouts in code

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  LinearLayout.LayoutParams lp;
  lp = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
                                     LayoutParams.FILL_PARENT);

  LinearLayout.LayoutParams textViewLP;
  textViewLP = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
                                             LayoutParams.WRAP_CONTENT);

  LinearLayout ll = new LinearLayout(this);
  ll.setOrientation(LinearLayout.VERTICAL);

  TextView myTextView = new TextView(this);
  myTextView.setText("Hello World, HelloWorld");
```

FIGURE 22 (Meier Book, Page 28)

Patent # 7,716,629 Claim Chart for the Android Framework

Meier Book, Page 209 defines how the "Content Providers" "decouples" data storage (content) from your application logic (function) on its third paragraph.



7

Databases and Content Providers

WHAT'S IN THIS CHAPTER?

- Creating databases and using SQLite
- Using Content Providers to share application data
- Querying Content Providers
- Using Cursors and Content Values to read and write from and to Content Providers
- Database design considerations
- Introduction to the native Content Providers
- Using the Contact Content Provider

In this chapter you'll be introduced to the SQLite library, and you'll look at how to use Content Providers to share and use structured data within and between applications.

SQLite offers a powerful SQL database library that provides a robust persistence layer over which you have total control.

Content Providers offer a generic interface to any data source by decoupling the data storage layer from the application layer.

By default, access to a database is restricted to the application that created it. Content Providers offer a standard interface your applications can use to share data with and consume data from other applications — including many of the native data stores.

INTRODUCING ANDROID DATABASES

Structured data persistence in Android is provided through the following mechanisms:

- **SQLite Databases** When managed, structured data is the best approach, Android offers the SQLite relational database library. Every application can create its own databases over which it has complete control.

FIGURE 23 (Meier Book, Page 209)

Patent # 7,716,629 Claim Chart for the Android Framework
Android's objects can be arbitrary objects. We created a sample application using the Android Software Developers' Kit (Android SDK). In the following example:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/textview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:text="@string/hello"/>
```

The Arbitrary Object "textview" is being called by its name (android:id="@+id/textview"), and additionally has three parameters.

When compiling and running the application, the compilation goes fine, and the application runs fine (it prints "Alo mundo,Hello Android!) on the Android device as seen in here"

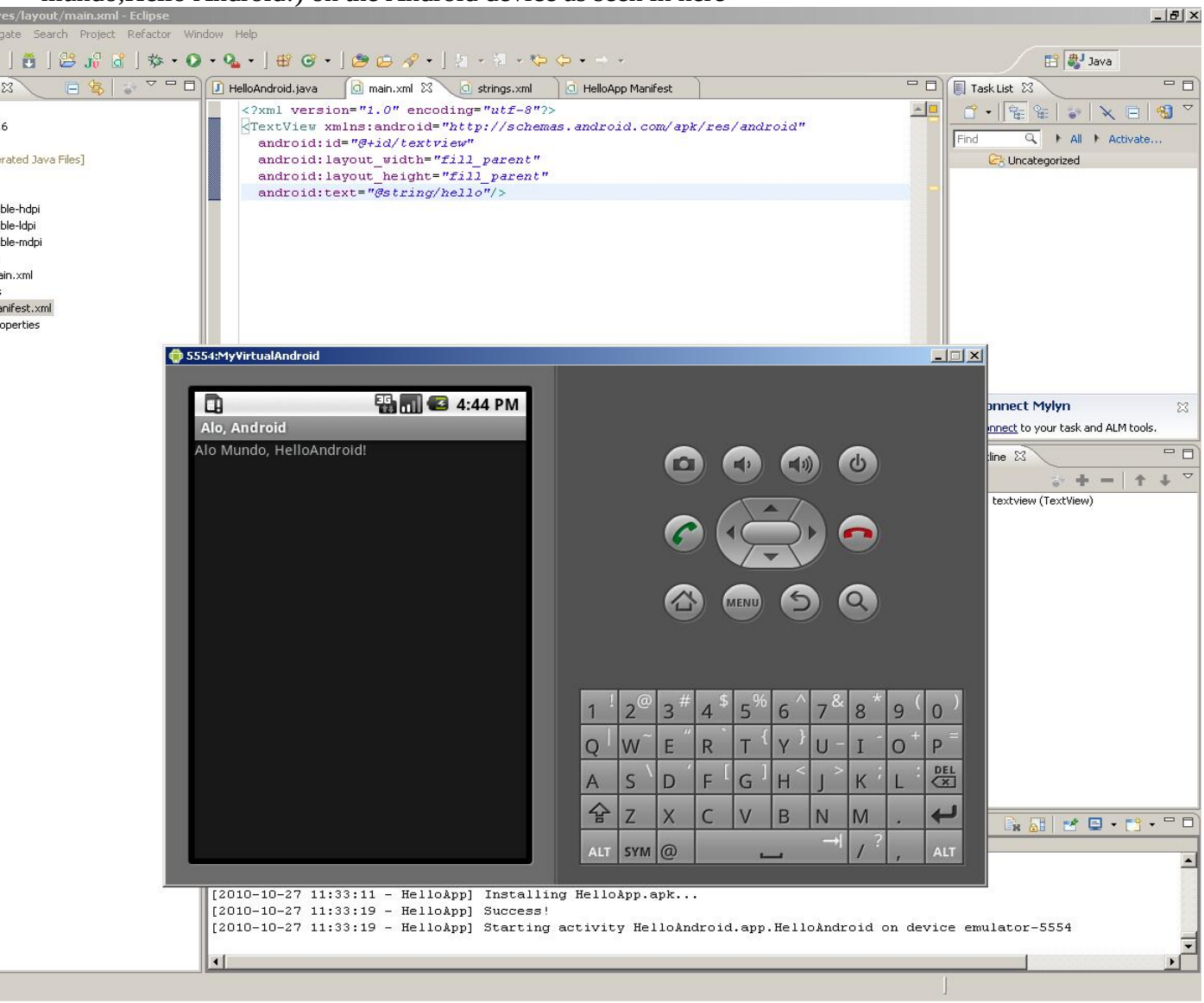


FIGURE 24 (Arbitrary Objects called with parameters)

However, when we modify the call to “textview” to include just the name of the Arbitrary Object, the call will look this way:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/textview"
  />
```

When making the above modification, the program compiles fine and the application runs normally, as can be seen on this screen print, we will call it “Sample Application”:

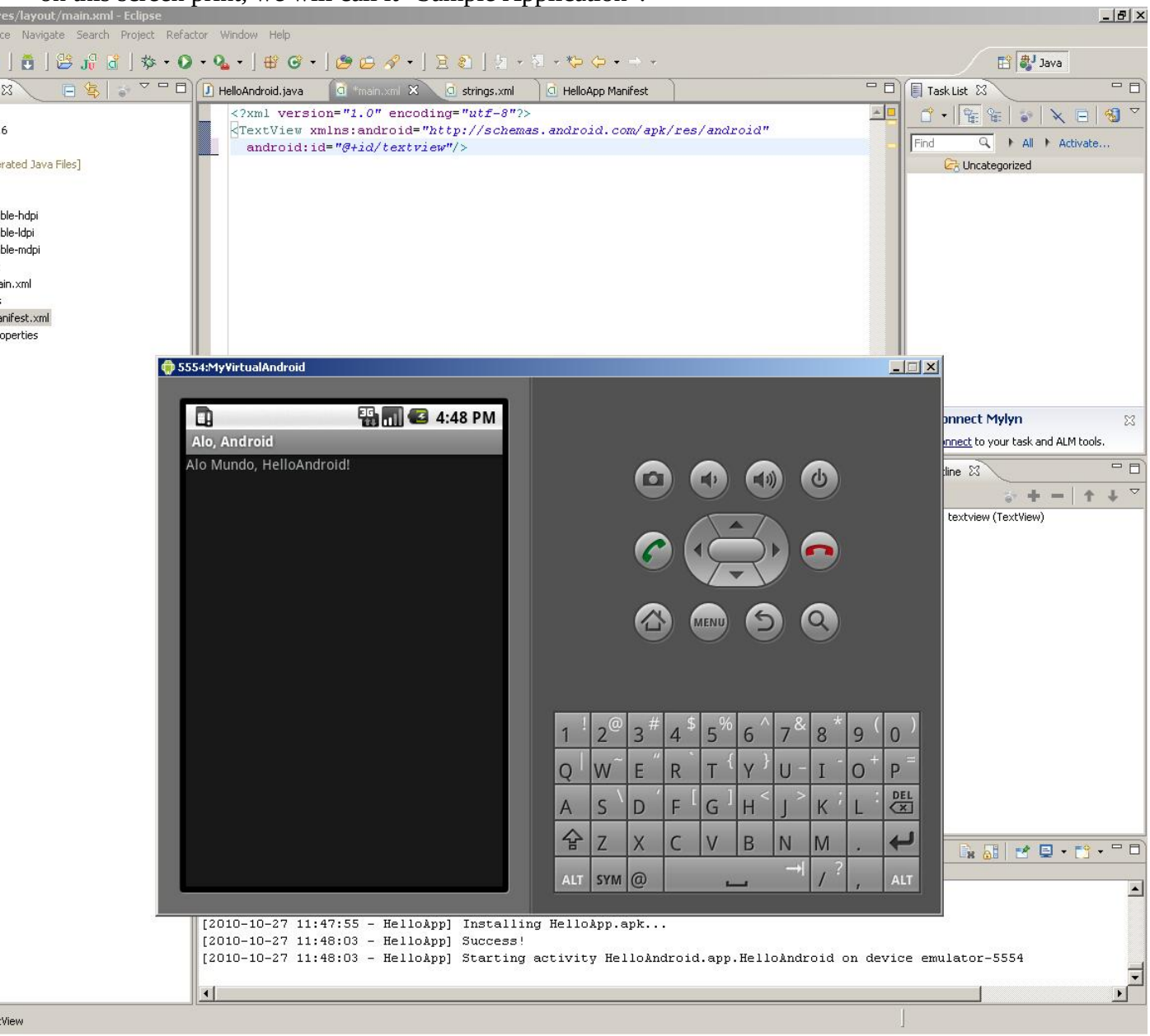


FIGURE 25 (Arbitrary Objects called by name only)

Claim 21, Element 1

“a first set of executable instructions for creating arbitrary objects with corresponding arbitrary names”

The creation of those Arbitrary objects can be using command line tools, such as a text editor, or using the recommended tool , which is named Eclipse. Using a text editor is a lot more work because all of the files needed to create the object need to be edited correctly and placed on the correct location (folders), in order for the compiler to work. Using Eclipse facilitates the process but it is not necessary to create the Arbitrary Objects.

The steps for generating a Computer application on the Android Framework when not using Eclipse are:

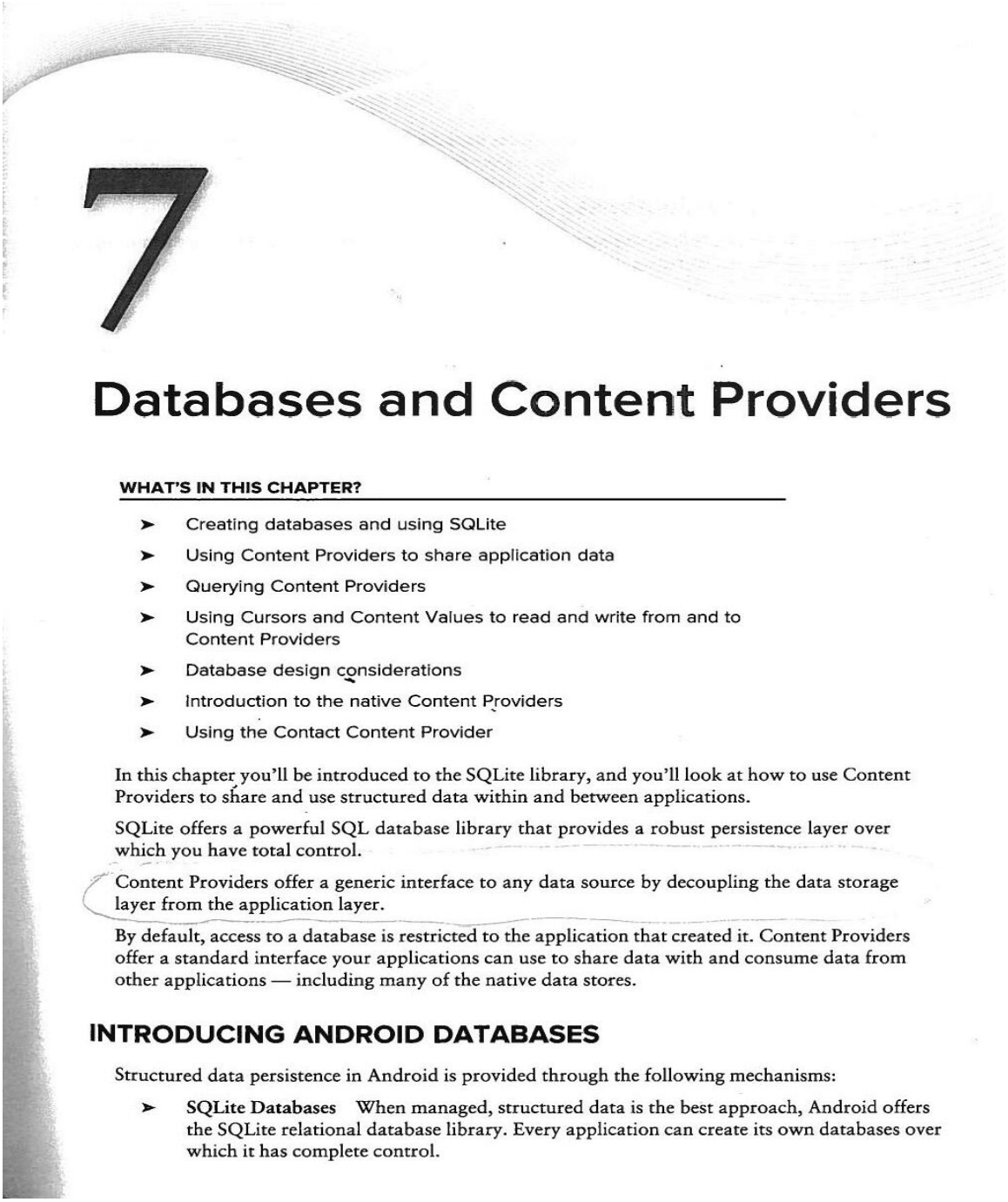
-Creation of the application either by manually creating each source file with a text editor
-Compiling of the application using a Java Compiler
-Creating a “package” (A file which’s extension is .APK”, using the Android SDK named apkbuilder.bat , which is distributed with the Android SDK.
-Moving that package file to the Android device, either by using Google’s “Android Market” application (in the case of a SmartPhone), or using the buil tin android application named “File Manager”, clicking on the file, and Android will then install it on the device.

When using eclipse, all is necessary is to click on the “Run” menu – that is what was done on the application described earlier.

Claim 21, Element 2

“of content objects used in generating said content of said computer application,”

The content can be generated by the “Content Provider” feature of Android. Meier Book, Pages 209-211 is a description of how it works. The title of the chapter, and throughout the whole text, it clarifies that the Android Framework can be used for generating the content:



7

Databases and Content Providers

WHAT'S IN THIS CHAPTER?

- Creating databases and using SQLite
- Using Content Providers to share application data
- Querying Content Providers
- Using Cursors and Content Values to read and write from and to Content Providers
- Database design considerations
- Introduction to the native Content Providers
- Using the Contact Content Provider

In this chapter you'll be introduced to the SQLite library, and you'll look at how to use Content Providers to share and use structured data within and between applications.

SQLite offers a powerful SQL database library that provides a robust persistence layer over which you have total control.

Content Providers offer a generic interface to any data source by decoupling the data storage layer from the application layer.

By default, access to a database is restricted to the application that created it. Content Providers offer a standard interface your applications can use to share data with and consume data from other applications — including many of the native data stores.

INTRODUCING ANDROID DATABASES

Structured data persistence in Android is provided through the following mechanisms:

- **SQLite Databases** When managed, structured data is the best approach, Android offers the SQLite relational database library. Every application can create its own databases over which it has complete control.

FIGURE 26 (Meier Book, Page 209)

- **Content Providers** Content Providers offer a generic, well-defined interface for using and sharing data.

Introducing SQLite Databases

Using SQLite you can create independent relational databases for your applications. Use them to store and manage complex, structured application data.

Android databases are stored in the `/data/data/<package_name>/databases` folder on your device (or emulator). By default all databases are private, accessible only by the application that created them.

Database design is a big topic that deserves more thorough coverage than is possible within this book. It is worth highlighting that standard database best practices still apply in Android. In particular, when you're creating databases for resource-constrained devices (such as mobile phones), it's important to normalize your data to reduce redundancy.

Introducing Content Providers

Content Providers provide an interface for publishing and consuming data, based around a simple URI addressing model using the `content://` schema. They let you decouple the application layer from the data layer, making your applications data-source agnostic by hiding the underlying data source.

Shared Content Providers can be queried for results, existing records updated or deleted, and new records added. Any application with the appropriate permissions can add, remove, or update data from any other application — including from the native Android databases.

Many native databases are available as Content Providers, accessible by third-party applications, including the phone's contact manager, media store, and other native databases as described later in this chapter.

By publishing your own data sources as Content Providers, you make it possible for you (and other developers) to incorporate and extend your data in new applications.

INTRODUCING SQLite

SQLite is a well regarded relational database management system (RDBMS). It is:

- Open-source
- Standards-compliant
- Lightweight
- Single-tier

It has been implemented as a compact C library that's included as part of the Android software stack.

By being implemented as a library, rather than running as a separate ongoing process, each SQLite database is an integrated part of the application that created it. This reduces external dependencies, minimizes latency, and simplifies transaction locking and synchronization.

SQLite has a reputation for being extremely reliable and is the database system of choice for many consumer electronic devices, including several MP3 players, the iPhone, and the iPod Touch.

FIGURE 27 (Meier Book, Page 210)

Lightweight and powerful, SQLite differs from many conventional database engines by loosely typing each column, meaning that column values are not required to conform to a single type. Instead, each value is typed individually for each row. As a result, type checking isn't necessary when assigning or extracting values from each column within a row.

For more comprehensive coverage of SQLite, including its particular strengths and limitations, check out the official site at `http://www.sqlite.org/`

CURSORS AND CONTENT VALUES

`ContentValues` are used to insert new rows into tables. Each Content Values object represents a single table row as a map of column names to values.

Queries in Android are returned as `Cursor` objects. Rather than extracting and returning a copy of the result values, Cursors are pointers to the result set within the underlying data. Cursors provide a managed way of controlling your position (row) in the result set of a database query.

The Cursor class includes a number of navigation functions including, but not limited to, the following:

- `moveToFirst` Moves the cursor to the first row in the query result
- `moveToNext` Moves the cursor to the next row
- `moveToPrevious` Moves the cursor to the previous row
- `getCount` Returns the number of rows in the result set
- `getColumnIndexOrThrow` Returns the index for the column with the specified name (throwing an exception if no column exists with that name)
- `getColumnName` Returns the name of the specified column index
- `getColumnNames` Returns a string array of all the column names in the current Cursor
- `moveToPosition` Moves the Cursor to the specified row
- `getPosition` Returns the current Cursor position

Android provides a convenient mechanism for simplifying the management of Cursors within your Activities. The `startManagingCursor` method integrates the Cursor's lifetime into the calling Activity's. When you've finished with the Cursor, call `stopManagingCursor` to do just that.

Later in this chapter you'll learn how to query a database and how to extract specific row/column values from the resulting Cursors.

WORKING WITH SQLite DATABASES

It's good practice to create a helper class to simplify your database interactions.

The following section shows you how to create a database adapter class for your database. This abstraction layer encapsulates your database interactions. It will provide intuitive, strongly typed methods for adding, removing, and updating items. A database adapter should also handle queries and expose methods for creating, opening, and closing the database.

FIGURE 28 (Meier Book, Page 211)

Claim 21, Element 3

"form objects used in defining said form of said computer application,"

Form in the Android Framework normally is defined on the Resource folder, referred to as "R" or "Layouts", having the default "layout.xml" as the Form, but this name can be changed. On Meier Book, Page 63 and 64 describes how they work. More specifically , the last three paragraphs of Page 63 define Layouts as your "Presentation Layer" and "User Interface", which is Form.


```
        <item name="attributeName">value</item>
    </style>
</resources>
```

Styles support inheritance using the `parent` attribute on the `<style>` tag, making it easy to create simple variations.

The following example shows two styles that can also be used as a theme: a base style that sets several text properties and a second style that modifies the first to specify a smaller font.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="BaseText">
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#111</item>
  </style>
  <style name="SmallText" parent="BaseText">
    <item name="android:textSize">8sp</item>
  </style>
</resources>
```

Drawables

Drawable resources include bitmaps and NinePatch (stretchable PNG) images. They also include complex composite Drawables, such as `LevelListDrawables` and `StateListDrawables` that can be defined in XML.

Both NinePatch Drawables and complex composite resources are covered in more detail in the next chapter.

All Drawables are stored as individual files in the `res/drawable` folder. The resource identifier for a Drawable resource is the lowercase file name without an extension.

> *The preferred format for a bitmap resource is PNG, although JPG and GIF files are also supported.*

Layouts

Layout resources let you decouple your presentation layer by designing user interface layouts in XML rather than constructing them in code.

The most common use of a layout is for defining the user interface for an Activity. Once defined in XML, the layout is "inflated" within an Activity using `setContentView`, usually within the `onCreate` method. You can also reference layouts from within other layout resources, such as layouts for each row in a List View. More detailed information on using and creating layouts in Activities can be found in Chapter 4.

Using layouts to create your screens is best-practice UI design in Android. The decoupling of the layout from the code lets you create optimized layouts for different hardware configurations, such as varying screen sizes, orientation, or the presence of keyboards and touchscreens.

FIGURE 29 (Meier Book, Page 63)

Each layout definition is stored in a separate file, each containing a single layout, in the `res/layout` folder. The file name then becomes the resource identifier.

A thorough explanation of layout containers and View elements is included in the next chapter, but as an example Listing 3-2 shows the layout created by the New Project Wizard. It uses a Linear Layout (described in more detail in Chapter 4) as a layout container for a Text View that displays the "Hello World" greeting.

LISTING 3-2: Hello World layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World!"
  />
</LinearLayout>
```

Animations

Android supports two types of animation. *Tweened animations* can be used to rotate, move, stretch, and fade a View; or you can create frame-by-frame animations to display a sequence of Drawable images. A comprehensive overview of creating, using, and applying animations can be found in Chapter 15.

Defining animations as external resources enables you to reuse the same sequence in multiple places and provides you with the opportunity to present different animations based on device hardware or orientation.

Tweened Animations

Each tweened animation is stored in a separate XML file in the project's `res/anim` folder. As with layouts and Drawable resources, the animation's file name is used as its resource identifier.

An animation can be defined for changes in `alpha` (fading), `scale` (scaling), `translate` (movement), or `rotate` (rotation).

Table 3-1 shows the valid attributes, and attribute values, supported by each animation type.

You can create a combination of animations using the `set` tag. An animation set contains one or more animation transformations and supports various additional tags and attributes to customize when and how each animation within the set is run.

The following list shows some of the set tags available.

- `duration` Duration of the animation in milliseconds.
- `startOffset` Millisecond delay before the animation starts.

FIGURE 30 (Meier Book, Page 64)

Patent # 7,716,629 Claim Chart for the Android Framework
Also on Meier Book, Page 50, it describes “Activities” as Presentation Layer (which can be Form) of Android Applications.

Mobile devices come in a large variety of shapes and sizes and are used across the world. In this chapter you'll learn how to externalize resources to ensure your applications run seamlessly on different hardware (particularly different screen resolutions and pixel densities), in different countries, and supporting multiple languages.

Next you'll examine the Application class, and learn how to extend it to provide a place for storing application state values.

Arguably the most important of the Android building blocks, the `Activity` class forms the basis for all your user interface screens. You'll learn how to create new Activities and gain an understanding of their life cycles and how they affect the application lifetime.

Finally, you'll be introduced to some of the Activity subclasses that simplify resource management for some common user interface components such as maps and lists.

WHAT MAKES AN ANDROID APPLICATION?

Android applications consist of loosely coupled components, bound by an application manifest that describes each component and how they all interact, as well as the application metadata including its hardware and platform requirements.

The following six components provide the building blocks for your applications:

- **Activities** Your application's presentation layer. Every screen in your application will be an extension of the `Activity` class. Activities use Views to form graphical user interfaces that display information and respond to user actions. In terms of desktop development, an Activity is equivalent to a Form. You'll learn more about Activities later in this chapter.
- **Services** The invisible workers of your application. Service components run in the background, updating your data sources and visible Activities and triggering Notifications. They're used to perform regular processing that needs to continue even when your application's Activities aren't active or visible. You'll learn how to create Services in Chapter 9.
- **Content Providers** Shareable data stores. Content Providers are used to manage and share application databases. They're the preferred means of sharing data across application boundaries. This means that you can configure your own Content Providers to permit access from other applications and use Content Providers exposed by others to access their stored data. Android devices include several native Content Providers that expose useful databases like the media store and contact details. You'll learn how to create and use Content Providers in Chapter 7.
- **Intents** An inter-application message-passing framework. Using Intents you can broadcast messages system-wide or to a target Activity or Service, stating your intention to have an action performed. The system will then determine the target(s) that will perform any actions as appropriate.
- **Broadcast Receivers** Intent broadcast consumers. If you create and register a Broadcast Receiver, your application can listen for broadcast Intents that match specific filter

FIGURE 31 (Meier Book, Page 50)

Claim 21, Element 4

"and function objects used in executing said functionality of said computer application,"

Meier Book, Page 50, second bullet, describes "Services" as "The invisible workers of the application", (Which can be Functionality):

Mobile devices come in a large variety of shapes and sizes and are used across the world. In this chapter you'll learn how to externalize resources to ensure your applications run seamlessly on different hardware (particularly different screen resolutions and pixel densities), in different countries, and supporting multiple languages.

Next you'll examine the Application class, and learn how to extend it to provide a place for storing application state values.

Arguably the most important of the Android building blocks, the `Activity` class forms the basis for all your user interface screens. You'll learn how to create new Activities and gain an understanding of their life cycles and how they affect the application lifetime.

Finally, you'll be introduced to some of the Activity subclasses that simplify resource management for some common user interface components such as maps and lists.

WHAT MAKES AN ANDROID APPLICATION?

Android applications consist of loosely coupled components, bound by an application manifest that describes each component and how they all interact, as well as the application metadata including its hardware and platform requirements.

The following six components provide the building blocks for your applications:

- **Activities** Your application's presentation layer. Every screen in your application will be an extension of the `Activity` class. Activities use Views to form graphical user interfaces that display information and respond to user actions. In terms of desktop development, an Activity is equivalent to a Form. You'll learn more about Activities later in this chapter.
- **Services** The invisible workers of your application. Service components run in the background, updating your data sources and visible Activities and triggering Notifications. They're used to perform regular processing that needs to continue even when your application's Activities aren't active or visible. You'll learn how to create Services in Chapter 9.
- **Content Providers** Shareable data stores. Content Providers are used to manage and share application databases. They're the preferred means of sharing data across application boundaries. This means that you can configure your own Content Providers to permit access from other applications and use Content Providers exposed by others to access their stored data. Android devices include several native Content Providers that expose useful databases like the media store and contact details. You'll learn how to create and use Content Providers in Chapter 7.
- **Intents** An inter-application message-passing framework. Using Intents you can broadcast messages system-wide or to a target Activity or Service, stating your intention to have an action performed. The system will then determine the target(s) that will perform any actions as appropriate.
- **Broadcast Receivers** Intent broadcast consumers. If you create and register a Broadcast Receiver, your application can listen for broadcast Intents that match specific filter

FIGURE 32 (Meier Book, Page 50)

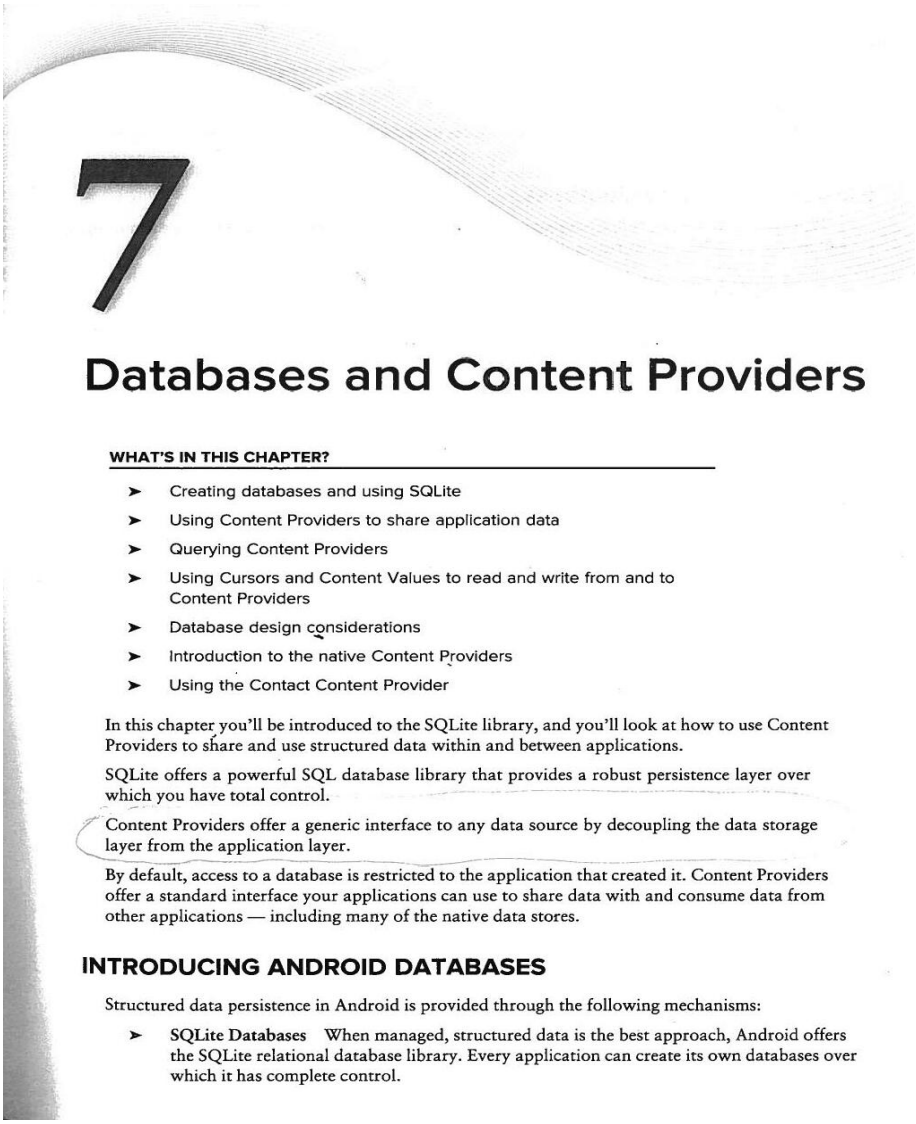
Claim 21, Element 5

“each arbitrary object being callable by name only,”
The Sample Application demonstrated in Claim 21, Element 1, calls the arbitrary object by name only.

Claim 21, Element 6

”each arbitrary object being independently modifiable without corresponding modifications being made to any other arbitrary object, “

In the Meier Book, there are many references of “decoupling” form from content, and from form, as a way of showing how these objects can be independently modified without changes to the other arbitrary objects. The Meier book, on page 209, defines how the “Content Providers” decouples data storage from your application logic on its third paragraph:



7

Databases and Content Providers

WHAT'S IN THIS CHAPTER?

- Creating databases and using SQLite
- Using Content Providers to share application data
- Querying Content Providers
- Using Cursors and Content Values to read and write from and to Content Providers
- Database design considerations
- Introduction to the native Content Providers
- Using the Contact Content Provider

In this chapter you'll be introduced to the SQLite library, and you'll look at how to use Content Providers to share and use structured data within and between applications.

SQLite offers a powerful SQL database library that provides a robust persistence layer over which you have total control.

Content Providers offer a generic interface to any data source by decoupling the data storage layer from the application layer.

By default, access to a database is restricted to the application that created it. Content Providers offer a standard interface your applications can use to share data with and consume data from other applications — including many of the native data stores.

INTRODUCING ANDROID DATABASES

Structured data persistence in Android is provided through the following mechanisms:

- **SQLite Databases** When managed, structured data is the best approach, Android offers the SQLite relational database library. Every application can create its own databases over which it has complete control.

FIGURE 33 (Meier Book, Page 209)

Patent # 7,716,629 Claim Chart for the Android Framework
The Meier Book, Page 63, under the headline named **"Layouts"**, defines how layouts decouple your user interface from your code:


```
        <item name="attributeName">value</item>
    </style>
</resources>
```

Styles support inheritance using the `parent` attribute on the `<style>` tag, making it easy to create simple variations.

The following example shows two styles that can also be used as a theme: a base style that sets several text properties and a second style that modifies the first to specify a smaller font.

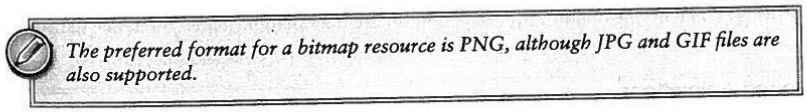
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="BaseText">
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#111</item>
  </style>
  <style name="SmallText" parent="BaseText">
    <item name="android:textSize">8sp</item>
  </style>
</resources>
```

Drawables

Drawable resources include bitmaps and NinePatch (stretchable PNG) images. They also include complex composite Drawables, such as `LevelListDrawables` and `StateListDrawables` that can be defined in XML.

Both NinePatch Drawables and complex composite resources are covered in more detail in the next chapter.

All Drawables are stored as individual files in the `res/drawable` folder. The resource identifier for a Drawable resource is the lowercase file name without an extension.


The preferred format for a bitmap resource is PNG, although JPG and GIF files are also supported.

Layouts

Layout resources let you decouple your presentation layer by designing user interface layouts in XML rather than constructing them in code.

The most common use of a layout is for defining the user interface for an Activity. Once defined in XML, the layout is "inflated" within an Activity using `setContentView`, usually within the `onCreate` method. You can also reference layouts from within other layout resources, such as layouts for each row in a List View. More detailed information on using and creating layouts in Activities can be found in Chapter 4.

Using layouts to create your screens is best-practice UI design in Android. The decoupling of the layout from the code lets you create optimized layouts for different hardware configurations, such as varying screen sizes, orientation, or the presence of keyboards and touchscreens.

FIGURE 34 (Meier Book, Page 63)

Claim 21, Element 7

“and each arbitrary object further being interchangeable with other arbitrary objects;”

Again in the Meier Book, Page 63 in the last paragraph it explains how these arbitrary objects are interchangeable, for the benefit of having the application support Android devices of varying screen sizes, keyboards and such:


```
        <item name="attributeName">value</item>
    </style>
</resources>
```

Styles support inheritance using the `parent` attribute on the `<style>` tag, making it easy to create simple variations.

The following example shows two styles that can also be used as a theme: a base style that sets several text properties and a second style that modifies the first to specify a smaller font.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="BaseText">
    <item name="android:textSize">14sp</item>
    <item name="android:textColor">#111</item>
  </style>
  <style name="SmallText" parent="BaseText">
    <item name="android:textSize">8sp</item>
  </style>
</resources>
```

Drawables

Drawable resources include bitmaps and NinePatch (stretchable PNG) images. They also include complex composite Drawables, such as `LevelListDrawables` and `StateListDrawables` that can be defined in XML.

Both NinePatch Drawables and complex composite resources are covered in more detail in the next chapter.

All Drawables are stored as individual files in the `res/drawable` folder. The resource identifier for a Drawable resource is the lowercase file name without an extension.

The preferred format for a bitmap resource is PNG, although JPG and GIF files are also supported.

Layouts

Layout resources let you decouple your presentation layer by designing user interface layouts in XML rather than constructing them in code.

The most common use of a layout is for defining the user interface for an Activity. Once defined in XML, the layout is “inflated” within an Activity using `setContentView`, usually within the `onCreate` method. You can also reference layouts from within other layout resources, such as layouts for each row in a List View. More detailed information on using and creating layouts in Activities can be found in Chapter 4.

Using layouts to create your screens is best-practice UI design in Android. The decoupling of the layout from the code lets you create optimized layouts for different hardware configurations, such as varying screen sizes, orientation, or the presence of keyboards and touchscreens.

FIGURE 35 (Meier Book, Page 63)

Claim 24

“The system of claim 21, wherein the third set of executable instructions are for deploying arbitrary objects locally”

The Sample Application demonstrated on Claim 21, Element 1, makes use of arbitrary objects locally.

Claim 28

“The system of claim 21, wherein the third set of executable instructions include instructions to access and deploy arbitrary objects into said design framework using said corresponding arbitrary names.”

The Android Framework enables access and deployment of said arbitrary names into said design framework.

Patent # 7,716,629 Claim Chart for the Android Framework

Claim 30

“The system of claim 21, further comprising executable instructions for caching objects.”

The Android Framework caches objects. In this picture showing the Android terminal emulation application, the response to the command “ls” shows a subdirectory named “cache”:

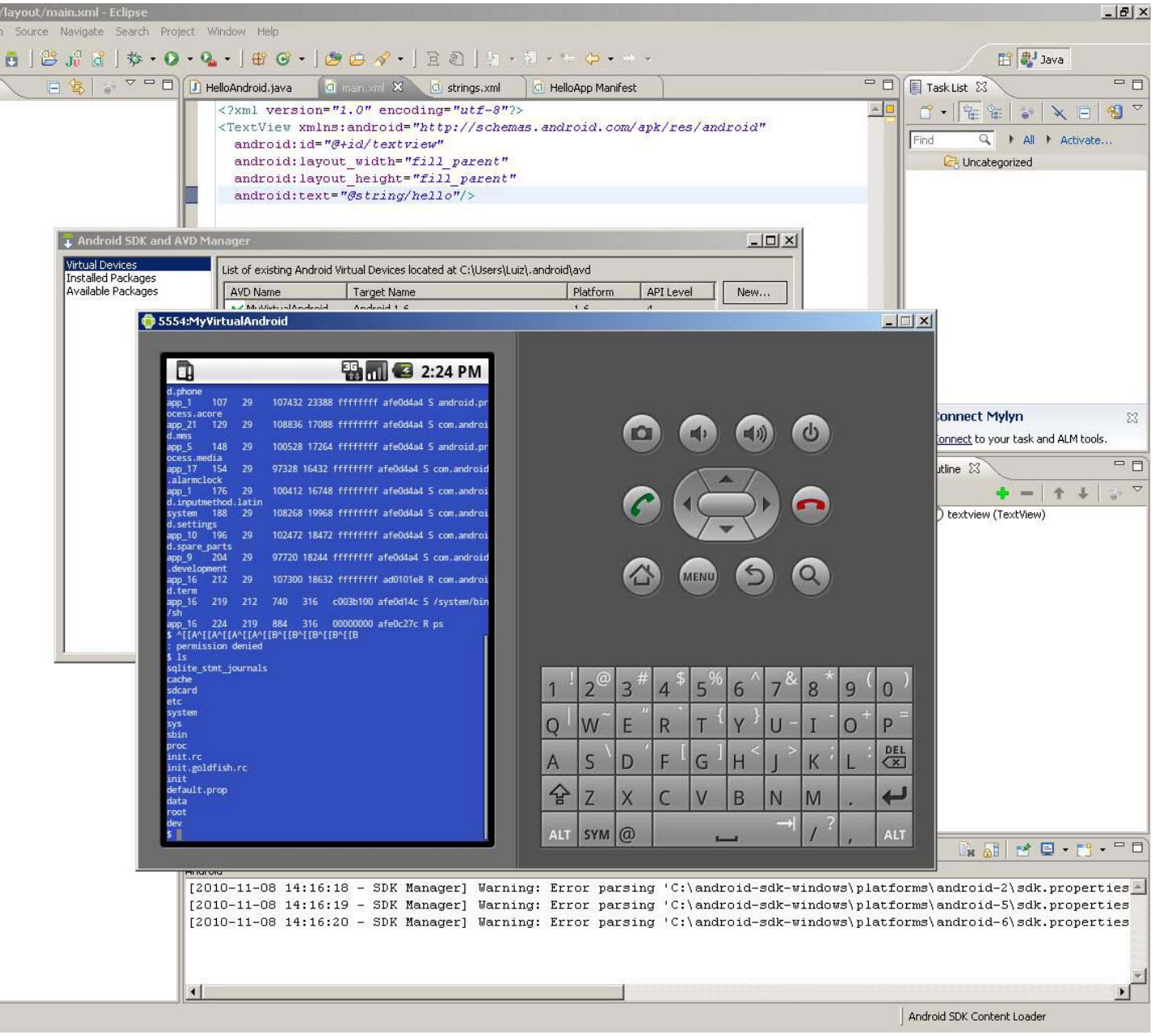


FIGURE 36 (use of cache)

Claim 32

“The system of claim 21, further comprising executable instructions for generating arbitrary objects in a programming language that is compatible and supported by said host system”

The Android Framework allows such arbitrary objects to be developed in Java, which is a programming language supported by the host system.