UNIT – II

Android Application Design

Mastering Important Android Terminology

Terminology used in Android application development

- ✓ **Context:** The context is the central command center for an Android application. All application-specific functionality can be accessed through the context.

- ✓ **Activity:** An Android application is a collection of tasks, each of which is called an Activity. Each Activity within an application has a unique task or purpose.

- ✓ **Intent:** The Android operating system uses an asynchronous messaging mechanism to match task requests with the appropriate Activity. Each request is packaged as an Intent.You can think of each such request as a message stating an intent to *do* something.

- ✓ **Service:** Tasks that do not require user interaction can be encapsulated in a service.A service is most useful when the operations are lengthy (offloading time-consuming processing) or need to be done regularly (such as checking a server for new mail).

Using the Application Context

The application Context is the central location for all top-level application functionality. The Context class can be used to manage application-specific configuration details as well as application-wide operations and data. Use the application Context to access settings and resources shared across multiple Activity instances.

1) Retrieving the Application Context

You can retrieve the Context for the current process using the

getApplicationContext() method, like this:

```
Context context = getApplicationContext();
```

2) Retrieving Application Resources

You can retrieve application resources using the getResources() method of the application Context. To retrieve a resource is by using its resource identifier, a unique number automatically generated within the R.java class.

1

The following example retrieves a String instance from the application resources by its resource ID:

```
String greeting getResources().getString(R.string.hello);
```

3) Accessing Application Preferences

You can retrieve shared application preferences using the getSharedPreferences() method of the application Context.

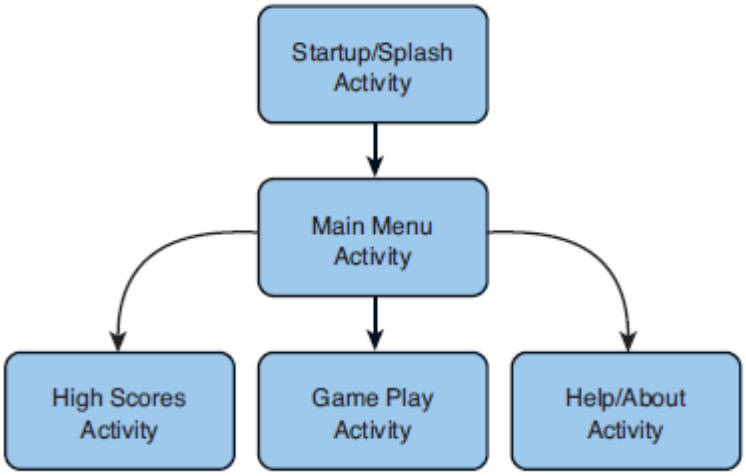
4) Accessing Other Application Functionality Using Context

The application Context provides access to a number of other top-level application features. Here are a few more things you can do with the application Context:

- ✓ Launch Activity instances
- ✓ Retrieve assets packaged with the application
- ✓ Request a system service (for example, location service)
- ✓ Manage private application files, directories, and databases
- ✓ Inspect and enforce application permissions

Performing Application Tasks with Activities

The Android Activity class (android.app.Activity) is core to any Android application. define and implement an Activity class for each screen in your application. For example, a simple game application might have the following five Activities,



- ✓ **A Startup or Splash screen:** This activity serves as the primary entry point to the application. It displays the application name and version information and transitions to the Main menu after a short interval.

- ✓ **A Main Menu screen:** This activity acts as a switch to drive the user to the core Activities of the application. Here the users must choose what they want to do within the application.

- ✓ **A Game Play screen:** This activity is where the core game play occurs.

- ✓ **A High Scores screen:** This activity might display game scores or settings.

- ✓ **A Help/About screen:** This activity might display the information the user might need to play the game.

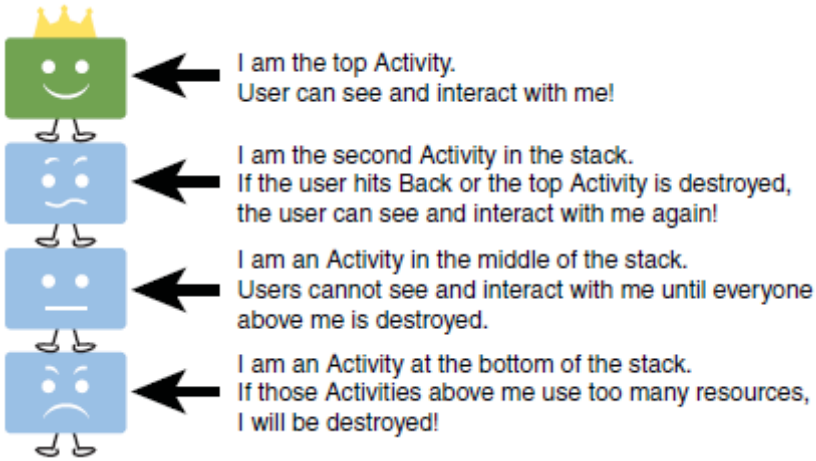
The Lifecycle of an Android Activity

Android applications can be multi-process, and the Android operating system allows multiple applications to run concurrently, provided memory and processing power.

Applications can have background processes, and applications can be interrupted and paused when events such as phone calls occur.

a single application Activity is in the foreground at any given time.

The Android operating system keeps track of all Activity objects running by placing them on an Activity stack (see Figure).When a new Activity starts, the Activity on the top of the stack (the current foreground Activity) pauses, and the new Activity pushes onto the top of the stack.When that Activity finishes, that Activity is removed from the activity stack, and the previous Activity in the stack resumes.

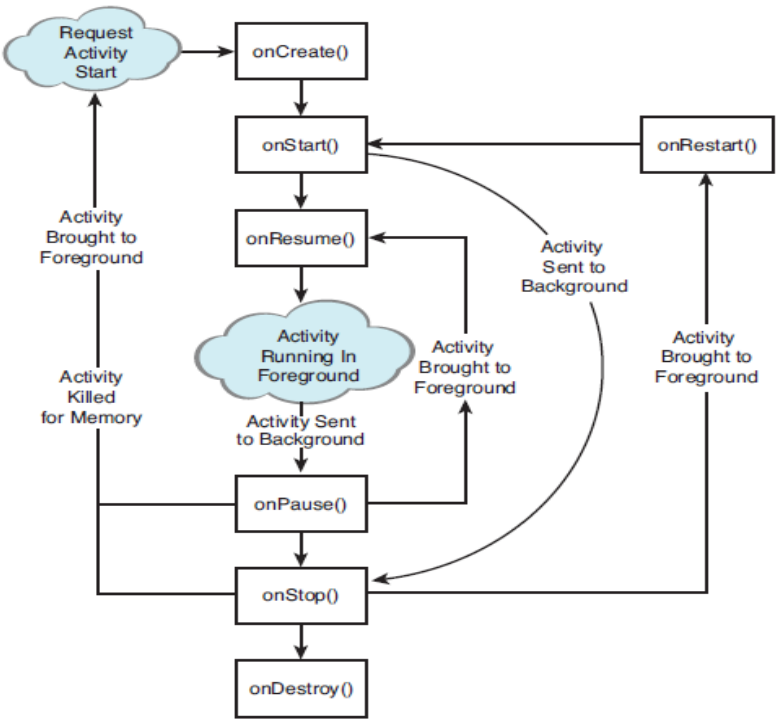


Android applications are responsible for managing their state and their memory, resources, and data.

Using Activity Callbacks to Manage Application State and Resources

Different important state changes within the Activity lifecycle are punctuated by a series of important method callbacks.

These callbacks are shown in below Figure



Here are the method stubs for the most important callbacks of the Activity class:

```
public class MyActivity extends Activity
{
protected void onCreate(Bundle savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
}
```

Now let's look at each of these callback methods, when they are called, and what they are used for.

Initializing Static Activity Data in onCreate()

- ✓ When an Activity first starts, the onCreate() method is called.

- ✓ onCreate() method has a single parameter, a Bundle, which is null if this is a newly started Activity.
- ✓ If this Activity was killed for memory reasons and is now restarted, the Bundle contains the previous state information for this Activity so that it can reinitiate.
- ✓ This includes calls to the setContentView() method.

Initializing and Retrieving Activity Data in onResume()

- ✓ When the Activity reaches the top of the activity stack and becomes the foreground process, the onResume() method is called.
- ✓ Although the Activity might not be visible yet to the user, this is the most appropriate place to retrieve any instances to resources (exclusive or otherwise) that the Activity needs to run.
- ✓ Often, these resources are the most process-intensive, so we only keep these around while the Activity is in the foreground.

Stopping, Saving, and Releasing Activity Data in onPause()

- ✓ When another Activity moves to the top of the activity stack, the current Activity is informed that it is being pushed down the activity stack by way of the onPause() method.
- ✓ Here, the Activity should stop any audio, video, and animations it started in the onResume() method. This is also where you must deactivate resources such as database
- ✓ Cursor objects if you have opted to manage them manually, as opposed to having them managed automatically.
- ✓ The onPause() method can also be the last chance for the Activity to clean up and release any resources it does not need while in the background

Avoiding Activity Objects Being Killed

- ✓ Under low-memory conditions, the Android operating system can kill the process for any Activity that has been paused, stopped, or destroyed. This essentially means that any Activity not in the foreground is subject to a possible shutdown.
- ✓ If the Activity is killed after onPause(), the onStop() and onDestroy() methods might not be called. The more resources released by an Activity in the onPause() method, the less likely the Activity is to be killed while in the background.
- ✓ **The act of killing an Activity does not remove it from the activity stack. Instead, the Activity state is saved into a Bundle object, assuming the Activity implements and uses onSaveInstanceState() for custom data, though some View data is automatically saved**.
- ✓ When the user returns to the Activity later, the onCreate() method is called again, this time with a valid Bundle object as the parameter.

Saving Activity State into a Bundle with onSaveInstanceState()

- ✓ If an Activity is vulnerable to being killed by the Android operating system due to low memory, the Activity can save state information to a Bundle object using the onSaveInstanceState() callback method.
- ✓ This call is not guaranteed under all circumstances, so use the onPause() method for essential data commits.
- ✓ When this Activity is returned to later, this Bundle is passed into the onCreate() method, allowing the Activity to return to the exact state it was in when the Activity paused.
- ✓ You can also read Bundle information after the onStart() callback method using the onRestoreInstanceState() callback.

Destroy Static Activity Data in onDestroy()

- ✓ When an Activity is being destroyed, the onDestroy() method is called. The onDestroy() method is called for one of two reasons:
  1) The Activity has completed its lifecycle voluntarily, or the Activity is being killed by the Android operating system because it needs the resources.
  2) Other Activity transitions are temporary, such as a child Activity displaying a dialog box, and then returning to the original Activity.
- ✓ In this case, the parent Activity launches the child Activity and expects a result.

Transitioning Between Activities with Intents

- ✓ A specific Activity can be designated as the main Activity to launch by default within the AndroidManifest.xml file
- ✓ Other Activities might be designated to launch under specific circumstances.
- ✓ **For example**, a music application might designate a generic Activity to launch by default from the Application menu, but also define specific alternative entry point Activities for accessing specific music playlists by playlist ID or artists by name.

Launching a New Activity by Class Name

- ✓ You can start activities in several ways.
- ✓ The simplest method is to use the Application Context object to call the startActivity() method, which takes a single parameter, an Intent.
- ✓ An Intent (android.content.Intent) is an asynchronous message mechanism used by the Android operating system to match task requests with the appropriate Activity or Service
- ✓ For example, calls the startActivity() method with an explicit Intent.This

Intent requests the launch of the target Activity named MyDrawActivity by its class. This class is implemented elsewhere within the package.

startActivity(new Intent(getApplicationContext(), MyDrawActivity.class));

Launching an Activity Belonging to Another Application

For example, a Customer Relationship Management (CRM) application might launch the Contacts application to browse the Contact database,
choose a specific contact, and return that Contact's unique identifier to the CRM application for use.

Here is an example of how to create a simple Intent with a predefined Action (ACTION_DIAL) to launch the Phone Dialer with a specific phone number to dial in the form of a simple Uri object:

Uri number Uri.parse(tel:5555551212);
Intent dial new Intent(Intent.ACTION_DIAL, number);
startActivity(dial);
You can find a list of commonly used Google application Intents at http://developer.android.com/guide/appendix/g-app-intents.html. Also available is the developer managed Registry of Intents protocols at OpenIntents, found at http://www.openintents.org/en/intentstable, which has a growing list of Intents available from third-party applications and those within the Android SDK.

Passing Additional Information Using Intents

You can also include additional data in an Intent.The Extras property of an Intent is stored in a Bundle object.The Intent class also has a number of helper methods for getting and setting name/value pairs for many common datatypes.

For example, the following Intent includes two extra pieces of information—a string value and a boolean:

Intent intent new Intent(this, MyActivity.class);
intent.putExtra("SomeStringData","Foo");
intent.putExtra("SomeBooleanData",false);

Working with Services

An Android Service is basically an Activity without a user interface. It can run as a background process or act much like a web service does,

- ✓ processing requests from third parties.You can use Intents and Activities to launch services using the startService() and bindService() methods. Any Services
- ✓ exposed by an Android application must be registered in the Android Manifest file.

- ✓ circumstances to use an Android service:
    1) A weather, email, or social network app might implement a service to routinely check for updates.
    2) A photo or media app that keeps its data in sync online might implement a service to package and upload new content in the background when the device is idle.
    3) A video-editing app might offload heavy processing to a queue on its service in order to avoid affecting overall system performance for non-essential tasks.
    4) A news application might implement a service to "pre-load" content by downloading news stories in advance of when the user launches the application, to improve performance.

Receiving and Broadcasting Intents

- ✓ Intents serve yet another purpose.You can broadcast an Intent object (via a call to **broadcastIntent())** to the Android system, and any application interested can receive that broadcast (called a **BroadcastReceiver**).Your application might do both sending of and listening for Intent objects.These types of Intent objects are generally used to inform the greater system that something interesting has happened and use special Intent
  Action types.

- ✓ For example, the Intent action ACTION_BATTERY_LOW broadcasts a warning when the battery is low. If your application is a battery-hogging Service of some kind, you might want to listen for this Broadcast and shut down your Service until the battery power is sufficient.

You can register to listen for battery/charge level changes by listening for the broadcast Intent object with the Intent action ACTION_BATTERY_CHANGED. There are also broadcast Intent objects for other interesting system events, such as SD card state changes, applications being installed or removed, and the wallpaper being changed

Configuring the Android Manifest File

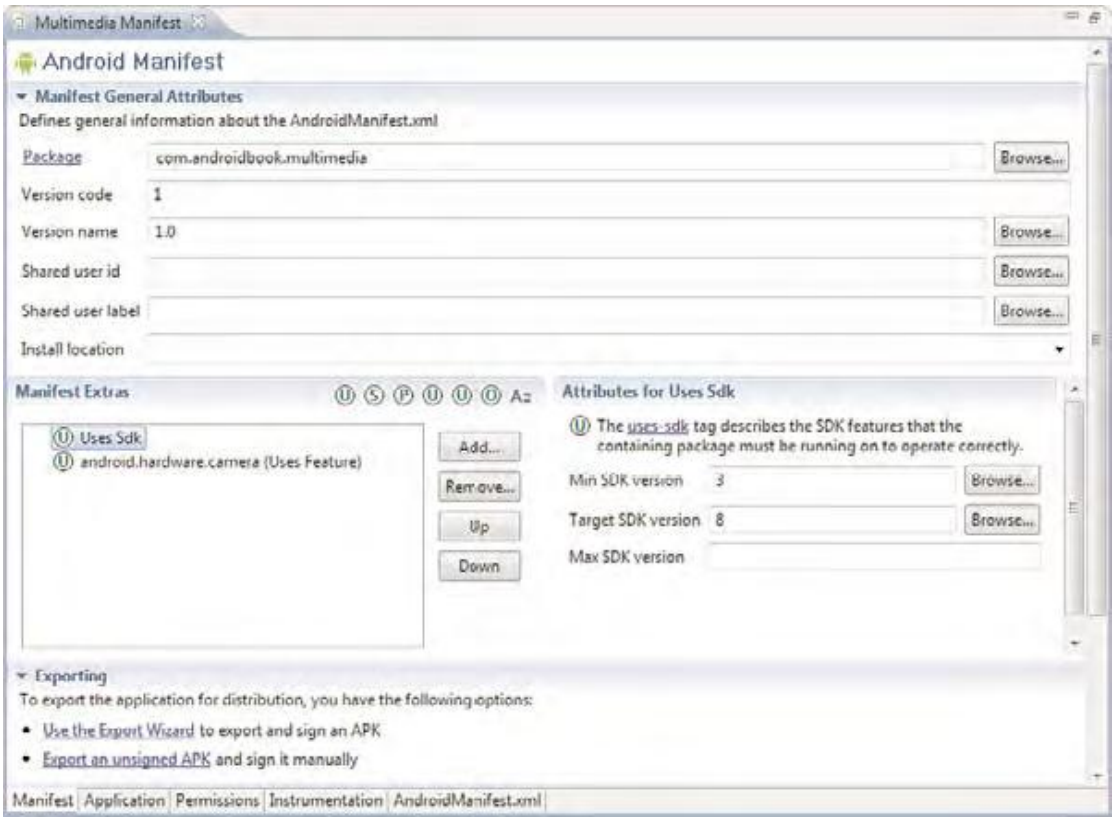
- ✓ The Android application manifest file is a specially formatted XML file that must accompany each Android application.This file contains important information about the application's identity.

- ✓ Here you define the application's name and version information and what application components the application relies upon, what permissions the application requires to run, and other application configuration information.
- ✓ The Android manifest file is named AndroidManifest.xml and must be included at
- ✓ the top level of any Android project.The information in this file is used by the Android system to
  1. Install and upgrade the application package.
  2. Display the application details such as the application name, description, and icon to users.
  3. Specify application system requirements, including which Android SDKs are supported, what hardware configurations are required (for example, d-pad navigation), and which platform features the application relies upon (for example, uses multitouch capabilities).
  4. Launch application activities.
  5. Manage application permissions.
  6. Configure other advanced application configuration details, including acting as a service, broadcast receiver, or content provider.
  7. Enable application settings such as debugging and configuring instrumentation for application testing.

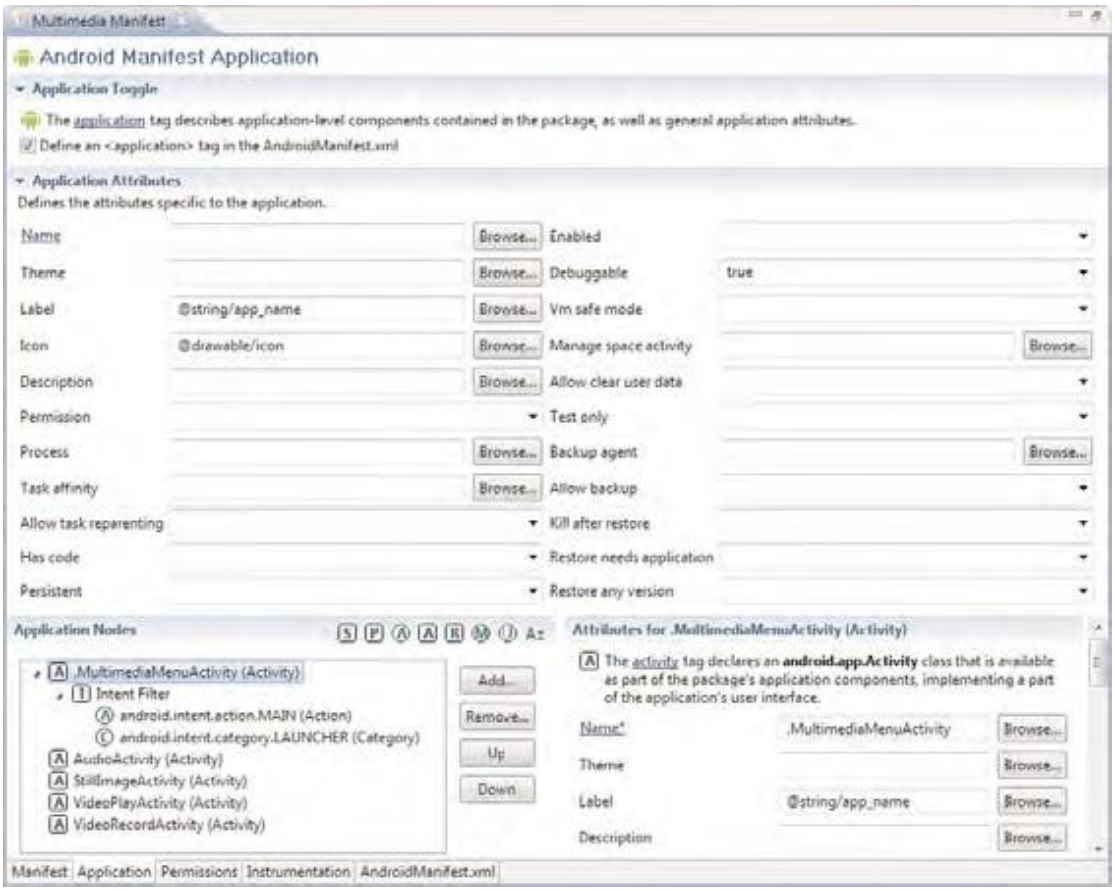
Editing the Manifest File Using Eclipse

- ✓ You can use the Eclipse Manifest File resource editor to edit the project manifest file. The Eclipse Manifest File resource editor organizes the manifest information into categories:

1. The Manifest tab
2. The Application tab
3. The Permissions tab
4. The Instrumentation tab
5. The AndroidManifest.xml tab



Configuring Package-Wide Settings Using the Manifest Tab
Managing Application and Activity Settings Using the Application Tab
Enforcing Application Permissions Using the Permissions Tab
Managing Test Instrumentation Using the Instrumentation Tab



The Application tab of the Eclipse Manifest File resource editor.

Android SDK Versions and Their API Levels

Android SDK Version	API Level (Value as Integer)
Android 1.0 SDK	1
Android 1.1 SDK	2
Android 1.5 SDK (Cupcake)	3
Android 1.6 SDK (Donut)	4
Android 2.0 SDK (Éclair)	5
Android 2.0.1 SDK (Éclair)	6
Android 2.1 SDK (Éclair)	7

What Are Resources?

- ✓ All Android applications are composed of two things: functionality (code instructions) and data (resources).The functionality is the code that determines how your application behaves.
- ✓ This includes any algorithms that make the application run. Resources include text strings, images and icons, audio files, videos, and other data used by the application.

Storing Application Resources

- ✓ Android resource files are stored separately from the java class files in the Android project. Most common resource types are stored in XML.You can also store raw data files and graphics as resources.

Understanding the Resource Directory Hierarchy

- ✓ Resources are organized in a strict directory hierarchy within the Android project. All resources must be stored under the /res project directory in specially named subdirectories that must be lowercase.
- ✓ Different resource types are stored in different directories.The resource sub-directories generated when you create an Android project using the Eclipse plug-in

Default Android Resource Directories

Resource Subdirectory	Purpose
/res/drawable-*/	Graphics Resources
/res/layout/	User Interface Resources
/res/values/	Simple Data such as Strings and Color Values, and so on

Each resource type corresponds to a specific resource subdirectory name. For example, all graphics are stored under the /res/drawable directory structure. Resources can be further organized in a variety of ways using even more specially named directory qualifiers.

For example, the /res/drawable-hdpi directory stores graphics for high-density screens, the /res/drawable-ldpi directory stores graphics for low-density screens, and the /res/drawable-mdpi directory stores graphics for medium-density screens. If you had a graphic resource that was shared by all screens, you would simply store that resource in the /res/drawabledirectory.We talk more about resource directory qualifiers

Resource Value Types

- ✓ Android applications rely on many different types of resources—such as text strings, graphics, and color schemes—for user interface design.
- ✓ These resources are stored in the /res directory of your Android project in a strict (but reasonably flexible) set of directories and files. All resources filenames must be lowercase and simple (letters, numbers, and underscores only).

- ✓ The resource types supported by the Android SDK and how they are stored within the project

Resource Type	**Required Directory**	**Filename**	**XML Tag**
Strings	/res/values/	strings.xml(suggested)	<string>
String Pluralization	/res/values/	strings.xml (suggested)	<plurals>, <item>
Arrays of Strings	/res/values/	strings.xml (suggested)	<string-array>,<item>
Booleans	/res/values/	bools.xml (suggested)	<bool>
Colors	/res/values/	Colors.xml (suggested)	<color>
Color State Lists	/res/color/	Examples include buttonstates.xml indicators.xml	<selector>, <item>
Dimensions	/res/values/	Dimens.xml (suggested)	<dimen>
Integers	/res/values/	integers.xml (suggested)	<integer>
Arrays of Integers	/res/values/	integers.xml (suggested)	<integer-array>,<item>
Mixed-Type Arrays	/res/values/	Arrays.xml (suggested)	<array>, <item>
Simple Drawables (Paintable)	/res/values/	drawables.xml (suggested)	<drawable>
Graphics	/res/drawable/	Examples include icon.png logo.jpg	Supported graphics files or drawabledefinition XML files such as shapes.
Tweened Animations	/res/anim/	Examples include fadesequence.xml spinsequence.xml	<set>, <alpha>, <scale>, <translate>, <rotate>
Frame-by-Frame Animations	/res/drawable/	Examples include sequence1.xml sequence2.xml	<animation-list>,<item>
Menus	/res/menu/	Examples include mainmenu.xml helpmenu.xml	<menu>
XML Files	/res/xml/	Examples include	Defined by the

		data.xml	developer
Raw Files	/res/raw/	Examples include jingle.mp3 somevideo.mp4 helptext.txt	Defined by the developer
Layouts	/res/layout/	Examples include main.xml help.xml	Varies. Must be a layout control
Styles and Themes	/res/values/	styles.xml themes.xml (suggested)	<style>

Storing Different Resource Value Types

All properly formatted files in the /res directory hierarchy and generates the class file R.java in your source code directory /src to access all variables.

Storing Simple Resource Types Such as Strings

Simple resource value types, such as strings, colors, dimensions, and other primitives, are stored under the /res/values project directory in XML files.

Storing Graphics, Animations, Menus, and Files

store numerous other types of resources, such as animation sequences, graphics, arbitrary XML files, and raw files.
For example,
you can include animation sequence definitions in the /res/anim directory. Make
sure you name resource files appropriately because the resource name is derived from the filename of the specific resource. For example, a file called flag.png in the /res/drawable directory is given the name R.drawable.flag.

- provide different logos for different screen densities by providing three versions of myLogo.png:
  - ✓ /res/drawable-ldpi/myLogo.png (low-density screens)
  - ✓ /res/drawable-mdpi/myLogo.png (medium-density screens)
  - ✓ /res/drawable-hdpi/myLogo.png (high-density screens)

- look much better if the layout was different in portrait versus landscape modes.We could change the layout around, moving controls around, in order to achieve a more pleasant user experience, and provide two layouts:
  - ✓ /res/layout-port/main.xml (layout loaded in portrait mode)
  - ✓ /res/layout-land/main.xml (layout loaded in landscape mode)
- If the device is in landscape mode, the layout in the /res/layout-land/main.xml is loaded. If it's in portrait mode, the /res/layout-port/main.xml layout is loaded.

Working with String Resources

String resources are among the simplest resource types available to the developer. String resources might show text labels on form views and for help text.The application name is also stored as a string resource, by default.
String resources are defined in XML under the /res/values project directory and
compiled into the application package at build time. All strings with apostrophes or single straight quotes need to be escaped or wrapped in double straight quotes. Some examples

Table 6.3 **String Resource Formatting Examples**

String Resource Value	Displays As
Hello, World	Hello, World
"User's Full Name:"	User's Full Name:
User\'s Full Name:	User's Full Name:
She said, \"Hi.\"	She said, "Hi."
She\'s busy but she did say, \"Hi.\"	She's busy but she did say, "Hi."

You can edit the strings.xml file using the Resources tab, or you can edit the XML directly by clicking the file and choosing the strings.xml tab.After you save the file, the resources are automatically added to your R.java class file.
String values are appropriately tagged with the <string> tag and represent a namevaluepair.The name attribute is how you refer to the specific string programmatically, so name these resources wisely.

example of the string resource file /res/values/strings.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
<string name ”app_name”>Resource Viewer</string>
<string name ”test_string”>Testing 1,2,3</string>
<string name ”test_string2”>Testing 4,5,6</string>
</resources>
```

a)Bold, Italic, and Underlined Strings

add three HTML-style attributes to string resources.These are bold, italic,
andunderlining.You specify the styling using the <b>, <i>, and <u> tags. For example

```
<string name ”txt”><b>Bold</b>,<i>Italic</i>,<u>Line</u></string>
```

b) Using String Resources Programmatically

```
String myStrHello
getResources().getString(R.string.hello);
```

other method:

```
CharSequencemyBoldStr
```

getResources().getText(R.string.boldhello);

2) Working with String Arrays

To store menu options and drop-down list values. String arrays are defined in XML under the /res/values project directory and compiled into the application package at build time.

String arrays are appropriately tagged with the <string-array> tag and a number of <item> child tags, one for each string in the array. Here's an example of a simple array resource

file /res/values/arrays.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>

<string-array name ”flavors”>
<item>Vanilla Bean</item>
<item>Chocolate Fudge Brownie</item>
<item>Strawberry Cheesecake</item>
<item>Coffee, Coffee, Buzz Buzz Buzz</item>
<item>Americone Dream</item>
</string-array>

<string-array name ”soups”>
<item>Vegetable minestrone</item>
<item>New England clam chowder</item>
<item>Organic chicken noodle</item>
</string-array>
</resources>
```

code retrieves a string array named flavors:

```
String[] aFlavors
getResources().getStringArray(R.array.flavors);
```

3) Working with Boolean Resources

Boolean resources can be used to store information about application game preferences and default values. Boolean resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

a) Defining Boolean Resources in XML

Boolean values are appropriately tagged with the <bool> tag and represent a name-value pair.The name attribute is how you refer to the specific Boolean value programmatically, so name these resources wisely.

Here's an example of the Boolean resource file /res/values/bools.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
```

<boolname ”bOnePlusOneEqualsTwo”>true</bool>
<boolname ”bAdvancedFeaturesEnabled”>false</bool>
</resources>

b) **Using Boolean Resources Programmatically**

To use a Boolean resource, you must load it using the Resource class.The following code accesses your application’s Boolean resource named bAdvancedFeaturesEnabled.

```
booleanbAdvancedMode
getResources().getBoolean(R.bool.bAdvancedFeaturesEnabled);
```

4) Working with Integer Resources

To strings and Boolean values, you can also store integers as resources. Integer resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

a) Defining Integer Resources in XML
   Integer values are appropriately tagged with the <integer> tag and represent a namevaluepair.The name attribute is how you refer to the specific integer programmatically, so name these resources wisely.

   Here’s an example of the integer resource file /res/values/nums.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
<integer name ”numTimesToRepeat”>25</integer>
<integer name ”startingAgeOfCharacter”>3</integer>
</resources>
```

b) Using Integer Resources Programmatically
   To use the integer resource, you must load it using the Resource class.
   The following code accesses your application’s integer resource named numTimesToRepeat:

```
intrepTimes
getResources().getInteger(R.integer.numTimesToRepeat);
```

5) Working with Colors

Android applications can store RGB color values, which can then be applied to other screen elements.You can use these values to set the color of text or other elements, such as the screen background. Color resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

RGB color values always start with the hash symbol (#).The alpha value can be given for transparency control.The following color formats are supported:

- ✓ #RGB (example, #F00 is 12-bit color, red)

- ✓ #ARGB (example, #8F00 is 12-bit color, red with alpha 50%)
- ✓ #RRGGBB (example, #FF00FF is 24-bit color, magenta)
- ✓ #AARRGGBB (example, #80FF00FF is 24-bit color, magenta with alpha 50%)

Color values are appropriately tagged with the <color> tag and represent a name-value pair. Here's an example of a simple color resource file /res/values/colors.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
<color name ”background_color”>#006400</color>
<color name ”text_color”>#FFE4C4</color>
</resources>
```

The example at the beginning of the chapter accessed a color resource. Color resources are simply integers.The following code retrieves a color resource called

```
prettyTextColor:
intmyResourceColor
        getResources().getColor(R.color.prettyTextColor);
```

6) Working with Dimensions

Many user interface layout controls such as text controls and buttons are drawn to specific dimensions.These dimensions can be stored as resources. Dimension values always end with a unit of measurement tag.

Dimension values are appropriately tagged with the <dimen> tag and represent a namevalue pair. Dimension resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

Unit of Measurement	Description	Resource Tag Required	Example
Pixels	Actual screen pixels	`px`	`20px`
Inches	Physical measurement	`in`	`1in`
Millimeters	Physical measurement	`mm`	`1mm`
Points	Common font measurement unit	`pt`	`14pt`
Screen density independent pixels	Pixels relative to 160dpi screen (preferable dimension for screen compatibility)	`dp`	`1dp`
Scale independent pixels	Best for scalable font display	`sp`	`14sp`

simple dimension resource file /res/values/dimens.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
```

<dimenname ”FourteenPt”>14pt</dimen>
<dimenname ”OneInch”>1in</dimen>
<dimenname ”TenMillimeters”>10mm</dimen>
<dimenname ”TenPixels”>10px</dimen></resources>
Dimension resources are simply floating point values.The following code retrieves a dimension resource called textPointSize:

```
floatmyDimension
        getResources().getDimension(R.dimen.textPointSize);
```

7) Working with Simple Drawables

specify simple colored rectangles by using the drawable resource type, which can then be applied to other screen elements.Thesedrawable types are defined in specific paint colors,much like the Color resources are defined.
Simple paintable drawable resources are defined in XML under the /res/values project directory and compiled into the application package at build time. Paintable drawable resources use the <drawable> tag and represent a name-value pair.

An example of a simple drawable resource file /res/values/drawables.xml:
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
<drawablename ”red_rect”>#F00</drawable>
</resources>

8) Working with Images

Applications often include visual elements such as icons and graphics.Android supports several image formats that can be directly included as resources for your application.These image formats

Supported Image Format	Description	Required Extension
Portable Network Graphics (PNG)	Preferred Format (Lossless)	.png
Nine-Patch Stretchable Images	Preferred Format (Lossless)	.9.png
Joint Photographic Experts Group (JPEG)	Acceptable Format (Lossy)	.jpg, .jpeg
Graphics Interchange Format (GIF)	Discouraged Format	.gif

9) Working with Animation

Android supports frame-by-frame animation and tweening. Frame-by-frame animation involves the display of a sequence of images in rapid succession.Tweened animation involves applying standard graphical transformations such as rotations and fades upon a single
image.
The Android SDK provides some helper utilities for loading and using animation resources.
These utilities are found in the android.view.animation.AnimationUtils class.

a) Defining and Using Frame-by-Frame Animation Resources

Frame-by-frame animation is often used when the content changes from frame to frame. This type of animation can be used for complex frame transitions—much like a kid's flip-book.
To define frame-by-frame resources, take the following steps:
1. Save each frame graphic as an individual drawable resource. It may help to name your graphics sequentially, in the order in which they are displayed—for example,
frame1.png, frame2.png, and so on.

2. Define the animation set resource in an XML file within /res/drawable/ resource directory.

3. Load, start, and stop the animation programmatically.

example of a simple frame-by-frame animation resource file
/res/drawable/juggle.xml that defines a simple three-frame animation that takes 1.5 seconds:

Frame-by-frame animation set resources defined with <animation-list> are represented by the Drawable subclass AnimationDrawable.The following code retrieves an Animation-Drawable resource called juggle:

```
importandroid.graphics.drawable.AnimationDrawable;
...
AnimationDrawablejugglerAnimation (AnimationDrawable)getResources().
getDrawable(R.drawable.juggle);
```

10) Working with Layouts

Much as web designers use HTML, user interface designers can use XML to define Android application screen elements and layout. A layout XML resource is where many different resources come together to form the definition of an Android application screen.
Layout resource files are included in the /res/layout/ directory and are compiled into the application package at build time. Layout files might include many user interface controls and define the layout for an entire screen or describe custom controls used in other layouts.
Here's a **simple example** of a layout file (/res/layout/main.xml) that sets the screen's background color and displays some text in the middle of the screen.

The main.xml layout file that displays this screen references a number of other resources, including colors, strings, and dimension values, all of which were defined in the strings.xml, colors.xml, and dimens.xml resource files.The color resource for the screen background color and resources for a TextView control's color, string, and text size follows:

```
<?xml version ”1.0” encoding ”utf-8”?>
<LinearLayout
xmlns:androidhttp://schemas.android.com/apk/res/androidandroid:orientation ”vertical”
android:layout_width ”fill_parent”
android:layout_height ”fill_parent”
android:background ”@color/background_color”>

<TextView
android:id ”@+id/TextView01”
android:layout_width ”fill_parent”
android:layout_height ”fill_parent”
android:text ”@string/test_string”
android:textColor ”@color/text_color”
android:gravity ”center”
android:textSize ”@dimen/text_size”>
</TextView>
</LinearLayout>
```

b) Defining and Using Tweened Animation Resources

Tweened animation features include scaling, fading, rotation, and translation.Theseactionscan be applied simultaneously or sequentially and might use different interpolators.
Tweened animation sequences are not tied to a specific graphic file, so you can write one sequence and then use it for a variety of different graphics. For example, you can make moon, star, and diamond graphics all pulse using a single scaling sequence, or you can make them spin using a rotate sequence.
Graphic animation sequences can be stored as specially formatted XML files in the /res/anim directory and are compiled into the application binary at build time.
Here's an example of a simple animation resource file /res/anim/spin.xml that defines a simple rotate operation—rotating the target graphic counterclockwise four times in place, taking 10 seconds to complete:

```
<?xml version ”1.0” encoding ”utf-8” ?>
<set xmlns:android
”http://schemas.android.com/apk/res/android”
android:shareInterpolator ”false”>
<set>
<rotate
```

```
android:fromDegrees ”0”
android:toDegrees ”-1440”
android:pivotX ”50%”
android:pivotY ”50%”
android:duration ”10000” />
</set>
</set>
```

If we go back to the example of a BitmapDrawableearlier,we can now add some animation simply by adding the following code to load the animation resource file spin.xml and set the animation in motion:

```
importandroid.view.animation.Animation;
importandroid.view.animation.AnimationUtils;
importandroid.widget.ImageView;
...
ImageViewflagImageView
(ImageView)findViewById(R.id.ImageView01);
flagImageView.setImageResource(R.drawable.flag);
...
Animation an
AnimationUtils.loadAnimation(this, R.anim.spin);
flagImageView.startAnimation(an);
```

Now you have your graphic spinning. Notice that we loaded the animation using the base class object Animation.You can also extract specific animation types using the subclasses that match: RotateAnimation, ScaleAnimation, TranslateAnimation, and
AlphaAnimation.
There are a number of different interpolators you can use with your tweened animation sequences.

Menus

You can also include menu resources in your project files. Like animation resources, menu resources are not tied to a specific control but can be reused in any menu control.
Each menu resource (which is a set of individual menu items) is stored as a specially formatted XML files in the /res/menu directory and are compiled into the application package at build time.
Here’s an example of a simple menu resource file /res/menu/speed.xml that defines a short menu with four items in a specific order:

```
<menu xmlns:android ”http://schemas.android.com/apk/res/android”>
<item
android:id ”@+id/start”
android:title ”Start!”
```

```
android:orderInCategory ”1”></item>
<item
android:id ”@+id/stop”
android:title ”Stop!”
android:orderInCategory ”4”></item>
<item
android:id ”@+id/accel”
android:title ”Vroom! Accelerate!”
android:orderInCategory ”2”></item>
<item
android:id ”@+id/decel”
android:title ”Decelerate!”
android:orderInCategory ”3”></item>
</menu>
```

You can create menus using the Eclipse plug-in, which can access the various configuration attributes for each menu item. In the previous case,we set the title (label) of each menu item and the order in which the items display. Now, you can use string resources for those titles, instead of typing in the strings. For example:

```
<menuxmlns:android “http://schemas.android.com/apk/res/android”>
<item
android:id ”@+id/start”
android:title ”@string/start”
android:orderInCategory ”1”></item>
<item
android:id ”@+id/stop”
android:title ”@string/stop”
android:orderInCategory ”2”></item>
</menu>
```

To access the preceding menu resource called /res/menu/speed.xml, simply override the method onCreateOptionsMenu() in your application:

```
publicbooleanonCreateOptionsMenu(Menu menu)
{
getMenuInflater().inflate(R.menu.speed, menu);
return true;
}
```

Layouts

Much as web designers use HTML, user interface designers can use XML to define Android application screen elements and layout.A layout XML resource is where many different resources come together to form the definition of an Android application screen.

Layout resource files are included in the /res/layout/ directory and are compiled into the application package at build time. Layout files might include many user interface controls and define the layout for an entire screen or describe custom controls used in other layouts.

Here's a simple example of a layout file (/res/layout/main.xml) that sets the screen's background color and displays some text in the middle of the screen.
The main.xml layout file that displays this screen references a number of other resources, including colors, strings, and dimension values, all of which were defined in the strings.xml, colors.xml, and dimens.xml resource files.The color resource for the screen background color and resources for a TextView control's color, string, and text size follows:

```
<?xml version ”1.0” encoding ”utf-8”?>
<LinearLayoutxmlns:android
“http://schemas.android.com/apk/res/android”
android:orientation ”vertical”
android:layout_width ”fill_parent”
android:layout_height ”fill_parent”
android:background ”@color/background_color”>
<TextView
android:id ”@+id/TextView01”
android:layout_width ”fill_parent”
android:layout_height ”fill_parent”
android:text ”@string/test_string”
android:textColor ”@color/text_color”
android:gravity ”center”
android:textSize ”@dimen/text_size”></TextView>
</LinearLayout>
```

Styles

- ✓ Android user interface designers can group layout element attributes together in styles.
- ✓ Layout controls are all derived from the View base class, which has many useful attributes.
- ✓ Individual controls, such as Checkbox, Button, and TextView, have specialized attributes associated with their behavior.
- ✓ Styles are tagged with the <style> tag and should be stored in the /res/values/ directory.
- ✓ Style resources are defined in XML and compiled into the application binary at build time.

example of a simple style resource file /res/values/styles.xml containing two styles: one for mandatory form fields, and one for optional form fields on TextView and EditText objects:
Many useful style attributes are colors and dimensions. It would be more appropriate to use references to resources. Here's the styles.xml file again; this time, the color and text size fields are available in the other resource files colors.xml and dimens.xml:

```
<?xml version ”1.0” encoding ”utf-8”?>
<resources>
<style name ”mandatory_text_field_style”>
<item name ”android:textColor”
>@color/mand_text_color</item>
<item name ”android:textSize”
>@dimen/important_text</item>
<item name ”android:textStyle”>bold</item>
</style>
<style name ”optional_text_field_style”>
<item name ”android:textColor”
>@color/opt_text_color</item>
<item name ”android:textSize”
>@dimen/unimportant_text</item>
<item name ”android:textStyle”>italic</item>
</style>
</resources>
```

form layout called /res/layout/form.xml

```
<?xml version ”1.0” encoding ”utf-8”?>
<LinearLayoutxmlns:android “http://schemas.android.com/apk/res/android”
android:orientation ”vertical”
android:layout_width ”fill_parent”
android:layout_height ”fill_parent”
android:background ”@color/background_color”>
<TextView
android:id ”@+id/TextView01”
style ”@style/mandatory_text_field_style”
android:layout_height ”wrap_content”
android:text ”@string/mand_label”
android:layout_width ”wrap_content” />
<EditText
android:id ”@+id/EditText01”
style ”@style/mandatory_text_field_style”
android:layout_height ”wrap_content”
android:text ”@string/mand_default”
android:layout_width ”fill_parent”
android:singleLine ”true” />
<TextView
```

```
android:id ”@+id/TextView02”
style ”@style/optional_text_field_style”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”@string/opt_label” />
<EditText
android:id ”@+id/EditText02”
style ”@style/optional_text_field_style”
android:layout_height ”wrap_content”
android:text ”@string/opt_default”
android:singleLine ”true”
android:layout_width ”fill_parent” />
<TextView
android:id ”@+id/TextView03”
style ”@style/optional_text_field_style”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”@string/opt_label” />
<EditText
android:id ”@+id/EditText03”
style ”@style/optional_text_field_style”
android:layout_height ”wrap_content”
android:text ”@string/opt_default”
android:singleLine ”true”
android:layout_width ”fill_parent” />
</LinearLayout>
```