Android Development Tutorial

New Android Lollipop Features

By Jim White

INTERTECH

Instructors Who Consult. | **Consultants** Who Teach.

Table of Contents:

Lollipop RecyclerView

READY FOR MATERIAL DESIGN

Material Design is a term you are going to hear a lot about in the new Android platform. Material Design is described as "a new approach for designing apps." Finding a clear and concise definition of Material Design can be daunting. Practically speaking, however, what Material Design brings is a new theme to be applied to apps, a new design guide which developers are encouraged to follow for UI layouts and components, widgets that can now have an elevation allowing them to cast a shadow, a new API to create custom animations such as the transition from on activity to another, and new View components for better and more flexible display of data.

RECYCLERVIEW

RecyclerView is "a more advanced and flexible version of ListView" (see here). In fact, at AnDevCon in November, I heard Chet Haase, Android UI Toolkit team lead at Google, describe RecyclerView as "ListView2."

BENEFITS OF USE

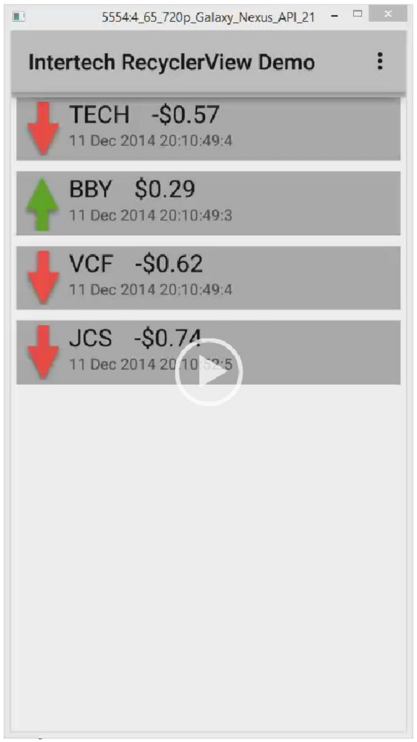
Why use RecyclerView? According to documentation, it scrolls more efficiently ("very efficiently") especially when considering large data sets and data sets whose elements are changing dynamically. RecyclerView also tries to simplify data display. In my opinion, there are more helper objects or pieces to building a RecyclerView, but default implementations are in place for many of the pieces and each of these associated components are typically small and simple – making it easier to know how to address RecyclerView features you want. The RecyclerView also offers some nice visual additions like item animation that is representative of the new Material Design.

RECYCLERVIEW TUTORIAL

Enough talk – you are probably ready to see and experiment with a RecyclerView. This RecyclerView tutorial assumes you have Android Studio installed on your PC and that your have used the Android SDK Manager to install Lollipop (API 21). Android Studio just emerged from beta this week. It is now “the official IDE for Android development” according to the developer.android.com web site. I am seeing a rather quick shift to Android Studio on the part of most Android developers and so I am going to use Android Studio on posts about Android on this blog site going forward. You can find the Android Studio project for this post on GitHub at https://github.com/IntertechInc/android-recycler.

The RecyclerView tutorial application here simulates a stock price ticker app – showing the rise and fall of a collection of financial stocks of interest. A background service and AsynTask simulate the constant fetch of new stock prices from Wall Street (for the tutorial the prices are actually generated by random number generator). The prices are going to be displayed on a RecyclerView. The data changes rapidly and there could be a lot of data – perfect requirements for this new Lollipop widget.

(click to see video)


5554:4_65_720p_Galaxy_Nexus_API_21

Intertech RecyclerView Demo

TECH -$0.57
11 Dec 2014 20:10:49:4

BBY $0.29
11 Dec 2014 20:10:49:3

VCF -$0.62
11 Dec 2014 20:10:49:4

JCS -$0.74
11 Dec 2014 20:10:52:5

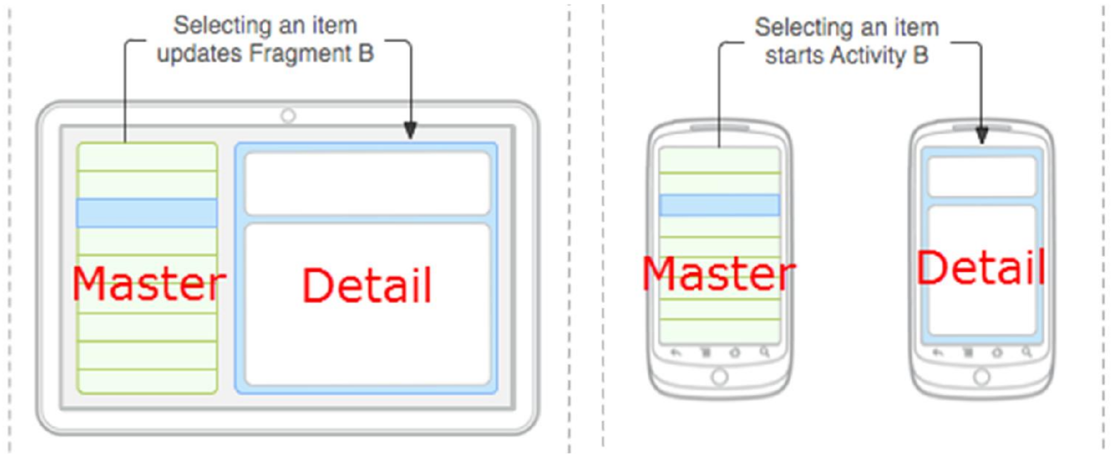
ADDING THE SUPPORT LIBRARY

Create a new Android project that has a Blank Activity with Fragment in Android Studio. [Again, you can download the Android Studio project from bitbucket if you want to review the code without writing it yourself.] RecyclerView is not part of the "normal" SDK. So you will need to add a support library to incorporate the RecyclerView (and its associated components) into your project. Add the RecyclerView (2nd compile line below) to your Gradle build file (build.gradle). Save the file, and clean/rebuild your project.

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  compile 'com.android.support:recyclerview-v7:21.0.0'
}
```

FRAGMENT LAYOUT WITH RECYCLERVIEW

Let's start building the app with the fragment layout that will contain the RecyclerView. We don't have to use a fragment – especially for this simple tutorial – but ever since Android 3, its always a good practice to provide the master list widget in a fragment so that you can easily create master/detail displays. Here the RecyclerView fragment provides the master list of stocks while a future second fragment could provide the detail information of any stock selected from the master list. The idea is that master and detail may be spread over a single activity screen when on a tablet or split across two activities when on a smaller smartphone device (see this tutorial for help on the ideas of master/detail displays in Android).

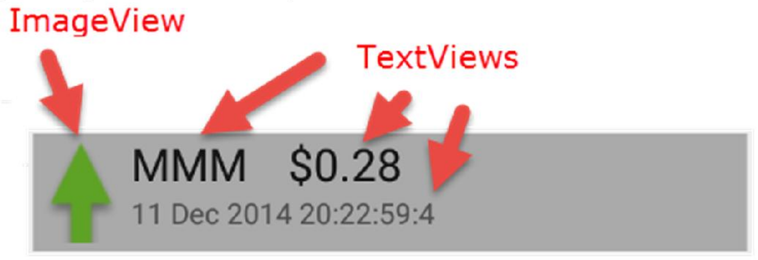

Selecting an item updates Fragment B
Master
Detail
Selecting an item starts Activity B
Master
Detail

The fragment, in this case, is comprised solely of the RecyclerView. Note the fully qualified widget reference.

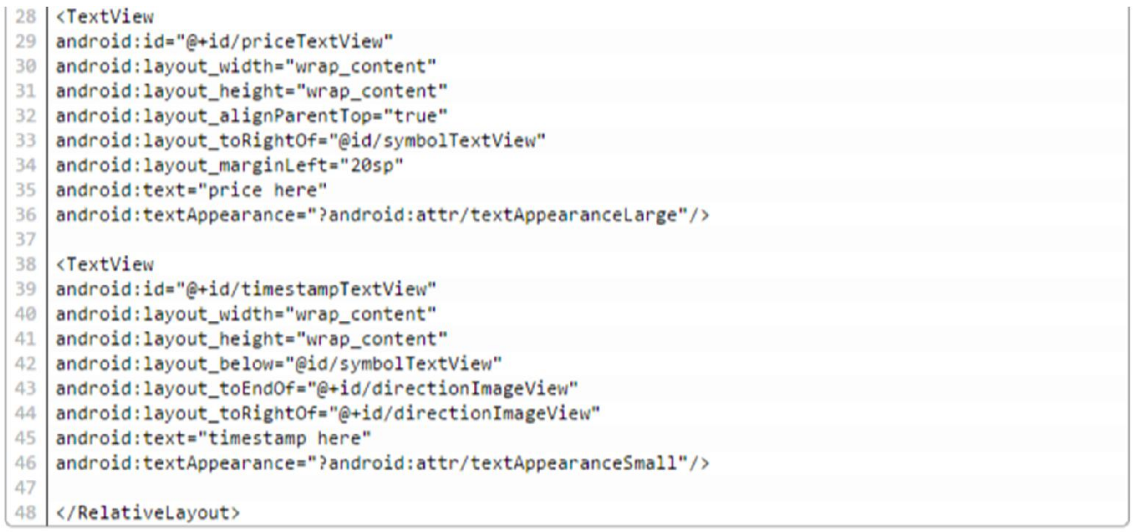
```
<android.support.v7.widget.RecyclerView
android:id="@+id/feedRecyclerView"
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity$PlaceholderFragment"/>
```

RECYCLERVIEW'S ROW LAYOUT

Next, we'll need a layout file to provide the arrangement of information in each row of the RecyclerView. The row layout will determine how each stock price will be displayed in the rows of the RecyclerView. This is not unlike how rows of a ListView had to have a layout. In this tutorial, each incoming stock price will be shown with a stock symbol, price change, timestamp of the price change and an image visually indicating whether the price change is a gain or a loss in price.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="5dp"
android:background="@android:color/darker_gray">

<ImageView
android:id="@+id/directionImageView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_alignParentTop="true"
android:padding="4sp"
android:src="@drawable/down"/>

<TextView
android:id="@+id/symbolTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentTop="true"
android:layout_toEndOf="@+id/directionImageView"
android:layout_toRightOf="@+id/directionImageView"
android:text="symbol here"
android:textAppearance="?android:attr/textAppearanceLarge"/>

```



```
<TextView
android:id="@+id/priceTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentTop="true"
android:layout_toRightOf="@id/symbolTextView"
android:layout_marginLeft="20sp"
android:text="price here"
android:textAppearance="?android:attr/textAppearanceLarge"/>

<TextView
android:id="@+id/timestampTextView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/symbolTextView"
android:layout_toEndOf="@+id/directionImageView"
android:layout_toRightOf="@+id/directionImageView"
android:text="timestamp here"
android:textAppearance="?android:attr/textAppearanceSmall"/>

</RelativeLayout>
```

To this point, easy-peasy and nothing different than what you have had to do to create a ListView to display similarly.

A NEW ADAPTER

Adapters in Android provide AdapterViews (like ListView) access to data and create the correct layout for each row or item in the AdapterView. However, RecyclerView does not use the old adapter hierarchy (BaseAdapter, SimpleAdapter, SimpleCursorAdapter, ArrayAdapter, etc.). Recycler view has its own RecyclerView.Adapter that replaces the old adapter. RecyclerView.Adapter is a generic type that requires you specify its ViewHolder via type parameters. ViewHolders are discussed in the next section.

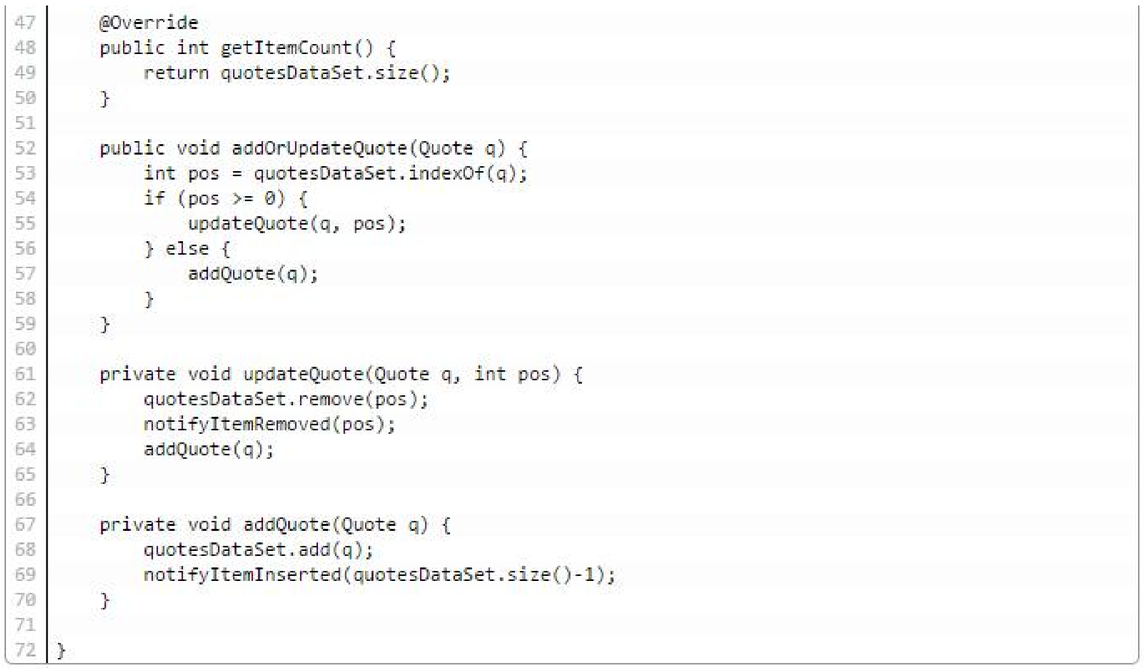
The new RecyclerView.Adapter abstract/generic class is similar in behavior to the old BaseAdapter, but it has its own unique interface. Here are the methods you need to implement:

- **getItemCount** – returns the number of rows in the associated data set and the number of rows that could be displayed in the RecyclerView.
- **onCreateViewHolder** – Called when the RecyclerView needs to create/add a new ViewHolder to represent a row. It creates the ViewHolder for the specified row display. Typically, this means inflating the row layout for the row and creating a ViewHolder with that layout.
- **onBindViewHolder** – Called by the RecyclerView when it needs to display the data at the position/row specified as a parameter. This method updates the row's View components with data from the dataset. Unlike the older AdapterViews (like ListView), the RecyclerView will not call this method again if the position of the item changes in the dataset unless the item itself is invalidate or a new position cannot be determined.

The RecyclerView.Adapter will need an associated data set. In this tutorial, a simple ArrayList of stock price objects is used. This list is updated by the background service and AsyncTask.

Here is the full blown RecyclerView.Adapter for the stock prices.

```java
package lollipop.intertech.com.recycler.recycler;

import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import java.util.List;

import lollipop.intertech.com.recycler.activity.R;
import lollipop.intertech.com.recycler.domain.Quote;
import lollipop.intertech.com.recycler.service.StockQuoteSimulator;

/**
 * Created by jwhite on 12/7/2014.
 */
public class QuoteAdapter extends RecyclerView.Adapter<QuoteViewHolder> {

    private static final int STARTING_QUOTES = 5;
    List<Quote> quotesDataSet;

    public QuoteAdapter() {
        quotesDataSet = (new StockQuoteSimulator()).initializeQuotes(STARTING_QUOTES);  // initial
    }

    @Override
    public QuoteViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        LayoutInflater inflater = LayoutInflater.from(viewGroup.getContext());
        View view = inflater.inflate(R.layout.quote_item, viewGroup, false);
        return new QuoteViewHolder(view);
    }

    @Override
    public void onBindViewHolder(QuoteViewHolder itemViewHolder, int i) {
        Quote quote = quotesDataSet.get(i);
        itemViewHolder.symbolTextView.setText(quote.getSymbol());
        itemViewHolder.priceTextView.setText(quote.getFormattedPrice());
        itemViewHolder.timestampTextView.setText(quote.getFormattedTimestamp());
        if (quote.getPrice() > 0) {
            itemViewHolder.directionImageView.setImageResource(R.drawable.up);
        } else {
            itemViewHolder.directionImageView.setImageResource(R.drawable.down);
        }
    }

```



```
    @Override
    public int getItemCount() {
        return quotesDataSet.size();
    }

    public void addOrUpdateQuote(Quote q) {
        int pos = quotesDataSet.indexOf(q);
        if (pos >= 0) {
            updateQuote(q, pos);
        } else {
            addQuote(q);
        }
    }

    private void updateQuote(Quote q, int pos) {
        quotesDataSet.remove(pos);
        notifyItemRemoved(pos);
        addQuote(q);
    }

    private void addQuote(Quote q) {
        quotesDataSet.add(q);
        notifyItemInserted(quotesDataSet.size()-1);
    }

}
```

Note too the notifyItemInserted( ) and notifyItemRemoved( ) methods used when a new quote is added or updated. There are a number of these types of methods on this new adapter type (notifyItemMoved, notifyItemChanged, etc.) and they can all be used conveniently to update the RecyclerView through the adapter.

VIEWHOLDER – OLD PATTERN, NEW IMPLEMENTATION

The RecyclerView.ViewHolder is also a new component (that follows a pre-Android 5 recommended pattern) associated with the new adapter and the RecyclerView. The ViewHolder's job is to cache the collection of row View objects. The findViewById( ) method to fetch View components gets expensive. In a ListView, this could happen a lot as the rows are continually rebuilt and redisplayed. The ViewHolder reduces that overhead.

In most case you simply extend the RecyclerView.ViewHolder class. As you can see by the code below, the ViewHolder gets each row's View components by id through the root view – itemView – passed to the constructor.

```
public class QuoteViewHolder extends RecyclerView.ViewHolder {

  protected TextView symbolTextView;
  protected TextView priceTextView;
  protected TextView timestampTextView;
  protected ImageView directionImageView;

  public QuoteViewHolder(View itemView) {
    super(itemView);
    directionImageView = (ImageView) itemView.findViewById(R.id.directionImageView);
    symbolTextView = (TextView) itemView.findViewById(R.id.symbolTextView);
    priceTextView = (TextView) itemView.findViewById(R.id.priceTextView);
    timestampTextView = (TextView) itemView.findViewById(R.id.timestampTextView);
    }
}
```

FRAGMENT AND LAYOUTMANAGER

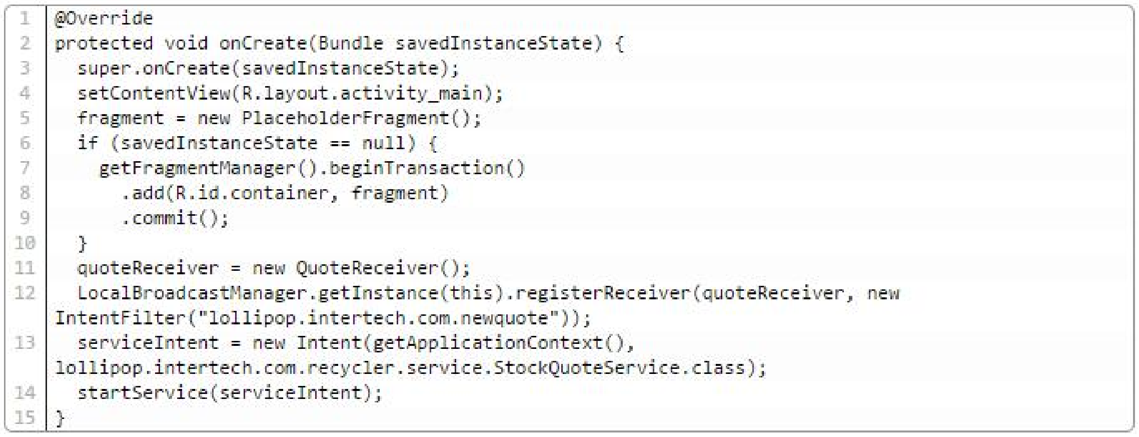
The RecyclerView is going to be displayed on a fragment. A hosting fragment's onCreateView( ) method is typically used to graphically initialize. Therefore, it is in the onCreateView( ) method of the fragment that the RecyclerView is initialized. You need a RecyclerView.LayoutManager to put items in the RecyclerView's child views. Unlike ListView or GridView, you can customize the layout of the child views (a point that will re reviewed below). The RecyclerView here uses a simple LinearLayoutManager provided by the support library. Below is the fragment's onCreateView( ) with the RecyclerView initializing code.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  View rootView = inflater.inflate(R.layout.fragment_main, container, false);
  quoteRecycler = (RecyclerView) rootView.findViewById(R.id.feedRecyclerView);
  quoteRecycler.setHasFixedSize(true);
  LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
  quoteRecycler.setLayoutManager(layoutManager);
  quoteAdapter = new QuoteAdapter();
  quoteRecycler.setAdapter(quoteAdapter);
  return rootView;
}
```

Beyond the setLayoutManager( ) call, and association of the RecyclerView to its RecyclerView.Adapter, take note of the setHasFixedSize( ) method. RecyclerView is about performance and display enhancements over widgets like ListView. In this case, use this method to help increase performance of the display when size of the RecyclerView is not going to change – even though the rows/items will come and go dynamically. As the documentation states: "RecyclerView can perform several optimizations if it can know in advance that changes in adapter content cannot change the size."

HOSTING ACTIVITY

The activity, in this demo, is there host the fragment holding the RecyclerView. It also starts/stops the stock price Service (and associate AsyncTask) and updates and forwards new price data from those to the fragment and RecyclerView. Here is the important onCreate( ) method.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  fragment = new PlaceholderFragment();
  if (savedInstanceState == null) {
    getFragmentManager().beginTransaction()
      .add(R.id.container, fragment)
      .commit();
  }
  quoteReceiver = new QuoteReceiver();
  LocalBroadcastManager.getInstance(this).registerReceiver(quoteReceiver, new
IntentFilter("lollipop.intertech.com.newquote"));
  serviceIntent = new Intent(getApplicationContext(),
lollipop.intertech.com.recycler.service.StockQuoteService.class);
  startService(serviceIntent);
}
```

You can peak at the rest of the activity code on GitHub.

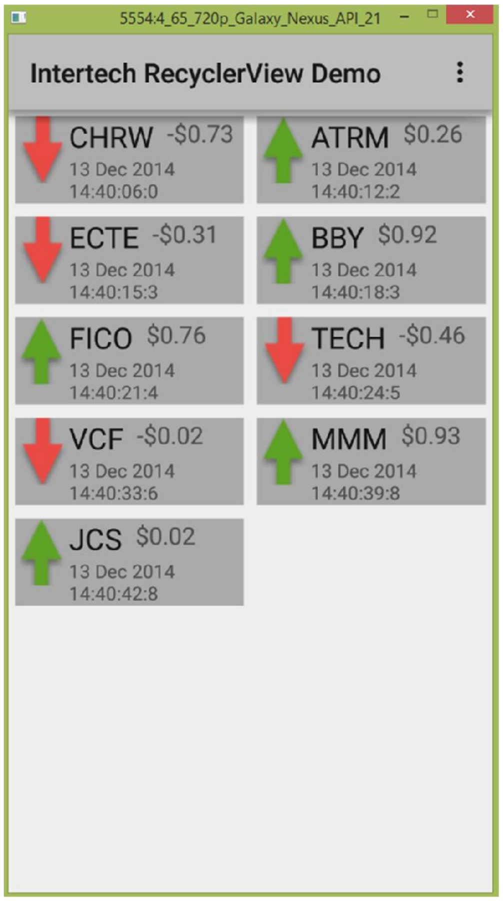
ADDITIONAL RECYCLERVIEW FEATURES

Beyond its performance improvements, RecyclerView comes with some display options that give it superior visual capability over the older components like ListView. Custom layout management and animation are two such capabilities.

LAYOUT MANAGEMENT

As mentioned above, when initializing the RecyclerView (see the hosting fragment's onCreateView( ) method), you need to create and set the LayoutManager for the RecyclerView. You can create your own custom layout or use LinearLayoutManager or GridLayoutManager (both subclasses of RecyclerView.LayoutManager) provided by the support library. Simply swap out the layout manager to the RecyclerView and you have a different display. Note the one line of code change below to use GridLayoutManager versus LinearLayoutManager.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  View rootView = inflater.inflate(R.layout.fragment_main, container, false);
  quoteRecycler = (RecyclerView) rootView.findViewById(R.id.feedRecyclerView);
  quoteRecycler.setHasFixedSize(true);
  //LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
  android.support.v7.widget.GridLayoutManager layoutManager = new
android.support.v7.widget.GridLayoutManager(getActivity(),2);
  quoteRecycler.setLayoutManager(layoutManager);
  quoteAdapter = new QuoteAdapter();
  quoteRecycler.setAdapter(quoteAdapter);
  return rootView;
}
```


5554:4_65_720p_Galaxy_Nexus_API_21
Intertech RecyclerView Demo

CHRW -$0.73
13 Dec 2014
14:40:06:0

ATRM $0.26
13 Dec 2014
14:40:12:2

ECTE -$0.31
13 Dec 2014
14:40:15:3

BBY $0.92
13 Dec 2014
14:40:18:3

FICO $0.76
13 Dec 2014
14:40:21:4

TECH -$0.46
13 Dec 2014
14:40:24:5

VCF -$0.02
13 Dec 2014
14:40:33:6

MMM $0.93
13 Dec 2014
14:40:39:8

JCS $0.02
13 Dec 2014
14:40:42:8

You can even use the layout manager to define the scrolling direction!

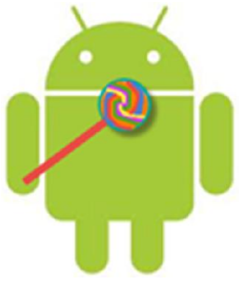
```
GridLayoutManager layoutManager = new GridLayoutManager(getActivity(),2,
GridLayoutManager.HORIZONTAL, true );
```

ANIMATION

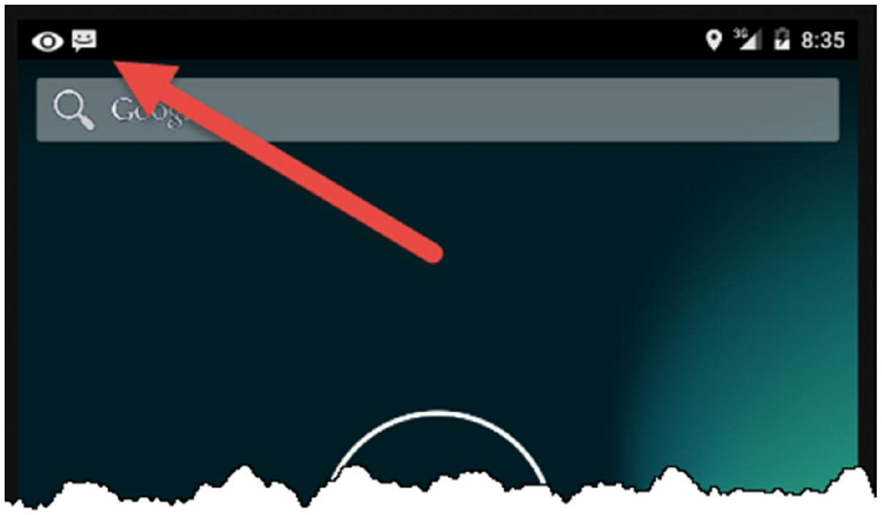
One of Lollipops considerable improvements is in the use of animation. Not surprising then that RecyclerView allows for custom item animation. You can determine how added, removed and update items get highlighted. As a minimal example of this capability, you can use the DefaultItemAnimator that is associated to the RecyclerView by default, but customize the add and remove animation duration time when you initialize the RecyclerView. This has the effect of exaggerating the coming and goings of the add and removed items of the view respectively.

```
quoteRecycler = (RecyclerView) rootView.findViewById(R.id.feedRecyclerView);
...
RecyclerView.ItemAnimator animator = quoteRecycler.getItemAnimator();
animator.setAddDuration(2000);
animator.setRemoveDuration(1000);
```

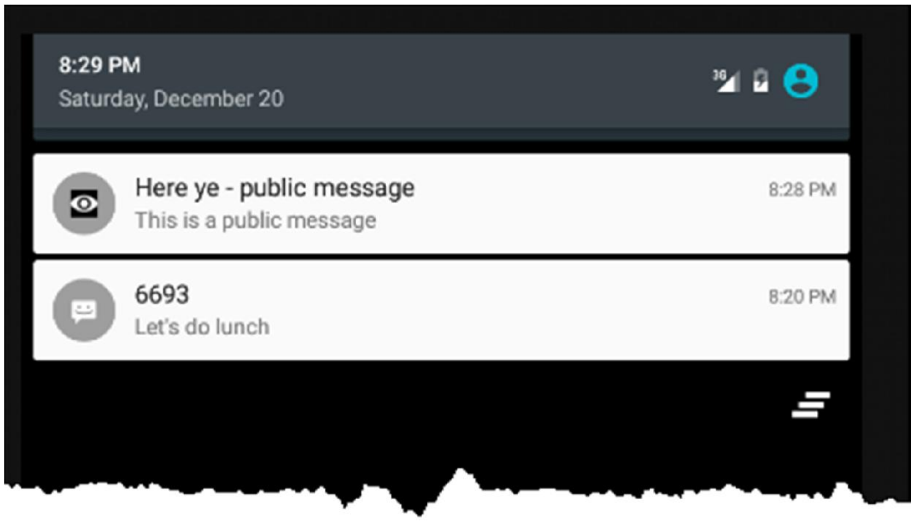
Lollipop Notifications



The first tutorial in our Android Development Tutorial series, based on new features in Android 5 Lollipop, we'll look at one of the new features that Android smartphone and tablet users will definitely see and appreciate – the Notification system. Notifications are displayed by an application in notification status bar (typically displayed on the top of smartphones and coming up from the bottom of a tablet).



A "notification drawer" can be pulled down to display more information about the notifications and to allow users to trigger application actions by pressing the notification in the drawer.


8:29 PM
Saturday, December 20

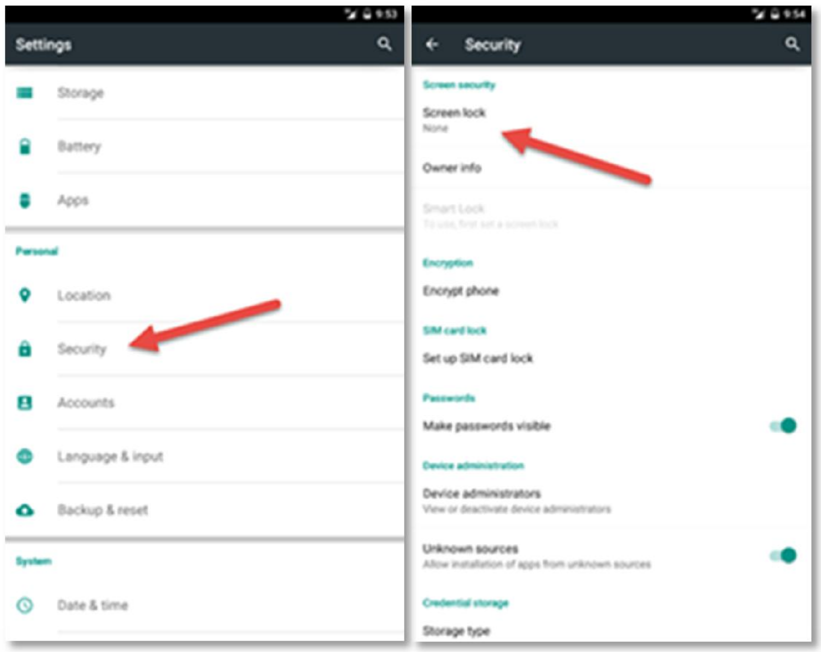
Here ye - public message 8:28 PM
This is a public message

6693 8:20 PM
Let's do lunch

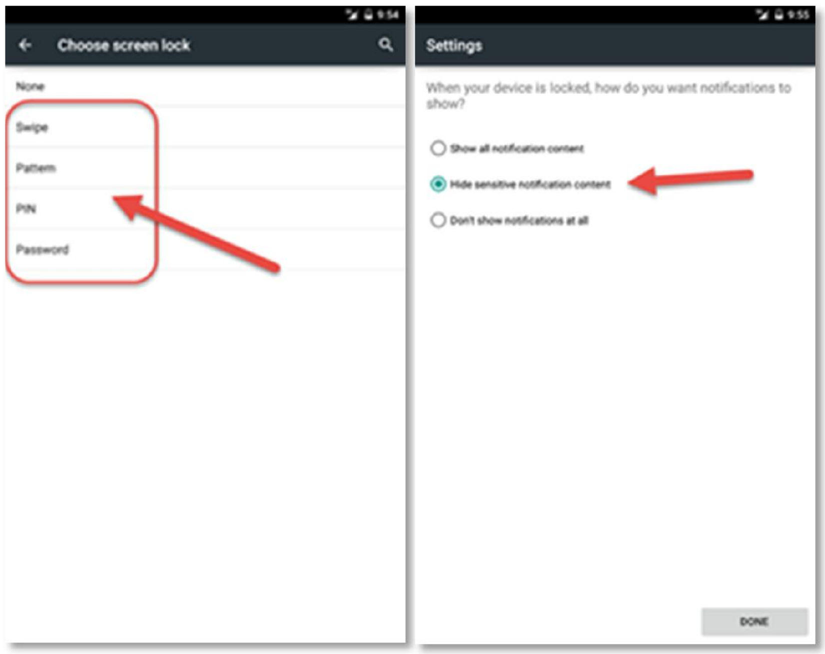
NOTIFICATIONS ON THE LOCK SCREEN

On Lollipop platforms, important notifications can be displayed on the lock screen. Users must configure their devices in order to display notifications on the lock screen. Users, or developers using AVD's, must take the following steps to enable lock screen notifications:

1. Find Settings and select Security and then Screen lock. You must first enable the screen lock in order to see the notifications on the screen lock.



2. Once you pick a screen lock, you'll get a chance to show or hide notifications. You can choose to show all of them, none of them, or show only the notifications that are not "sensitive." The definition of sensitive will be defined below.



Choose screen lock

None

Swipe

Pattern

PIN

Password

Settings

When your device is locked, how do you want notifications to show?

( ) Show all notification content

(●) Hide sensitive notification content

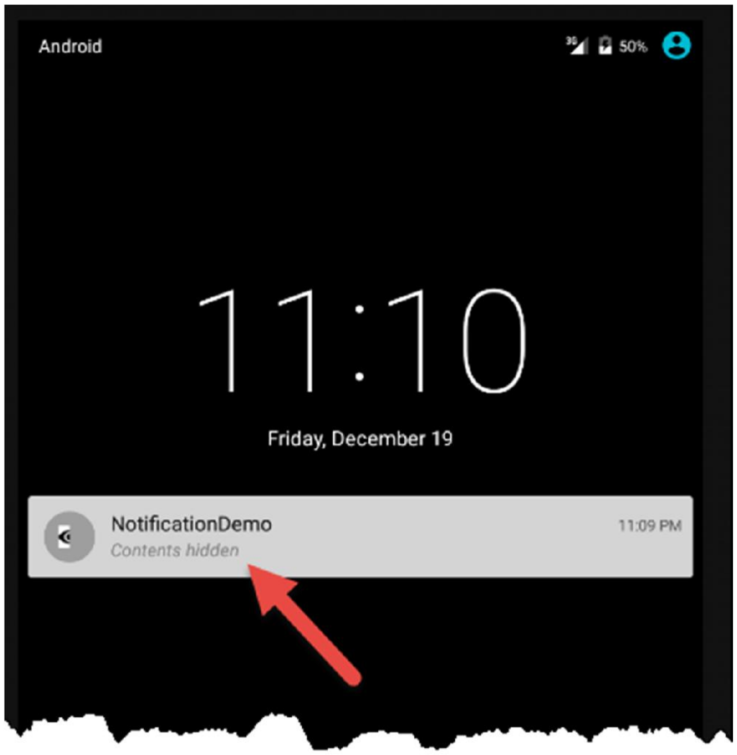
( ) Don't show notifications at all

By the way, testing your app on an AVD, you invoke the lock screen by pressing F7 on your keyboard.

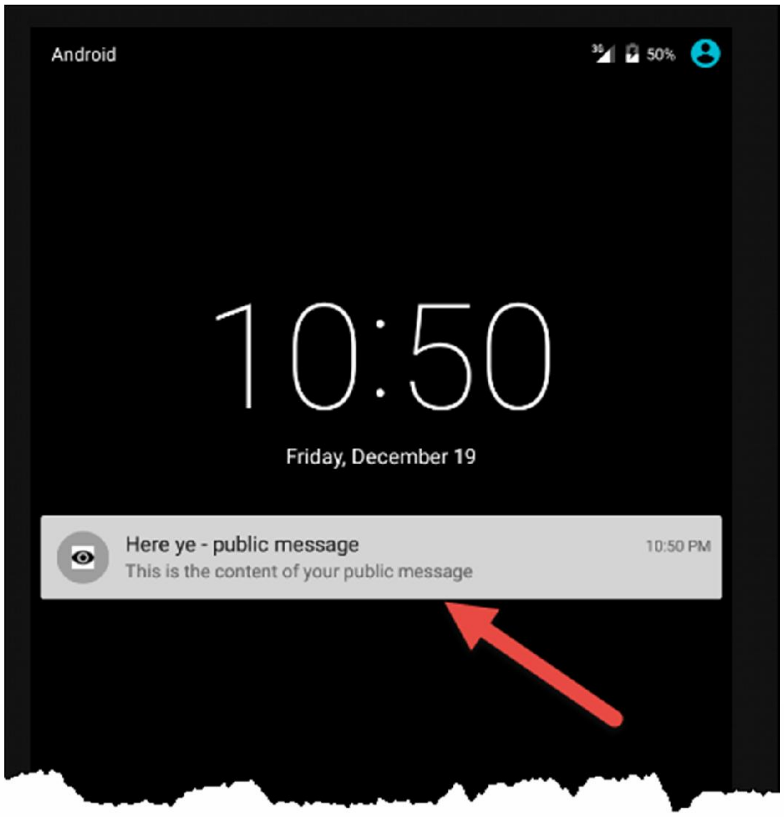
NOTIFICATION VISIBILITY

Developers can now (with Android 5) determine whether a notification message is "sensitive." Sensitivity is actually determined by a new Android 5 attribute called "visibility" on the notification object. There are three choices of visibility:

- Secret (Notification.VISIBILITY_SECRET) – messages are considered sensitive and not shown on the lock screen.
- Private (VISIBILITY_PRIVATE) – messages are considered sensitive such that the content is not displayed on the lock screen, but the fact that notification has arrived is displayed on the lock screen. Notice in the picture below that the lock screen displays that a private notification has been posted (from NotificationDemo app with its choice of icon), but the content of the message is still hidden and replaced by "*Contents hidden*".



- Public (VISIBILITY_PUBLIC) – these notification messages are displayed – to include the content – on the lock screen. As shown by the example below, now the notification title, contents, and icon are all on display on the lock screen.



When creating a notification message, developers use the setVisibility(int) method to select the visibility from public static final int options above. Below is code that sets the visibility to **public** (notifications are private by default). A little sample application to see this code exercised is available at GitHub. [For a full tutorial on Notification and NotificationManager API see here.]

```
Notification notification  = new Notification.Builder(this)
  .setCategory(Notification.CATEGORY_MESSAGE)
  .setContentTitle(title)
  .setContentText(text)
  .setSmallIcon(icon)
  .setAutoCancel(true)
  .setVisibility(visibility).build();
NotificationManager notificationManager =
  (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
notificationManager.notify(notification_id, notification );
```

An additional method on the builder – .setPublicVersion(notification) – allows you to provide a replacement notification to display on the lock screen when the visibility is set to private and you still want something to display on the lock screen.

BACKWARD COMPATIBILITY

If you intend to support older devices (pre-Lollipop and API 21), you will want to use NotificationCompat versus Notification to build the notification object.

```
Notification notification  = new NotificationCompat.Builder(this)
```

Speaking of backward compatibility support, remember notification visibility is an Android 5 feature. Therefore, unless you are creating an application just for API 21 devices (of which there are very few at this time), you need to add the support library to your application. If using Android Studio - the newly "official IDE" for Android development, you need to add the support library to the dependencies in the app's Gradle build file (build.gradle) as shown below (copy the second dependency).

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  compile 'com.android.support:appcompat-v7:21.0.+'
}
```

Keep in mind, older devices will not show the notifications on the lock screen (regardless of visibility).

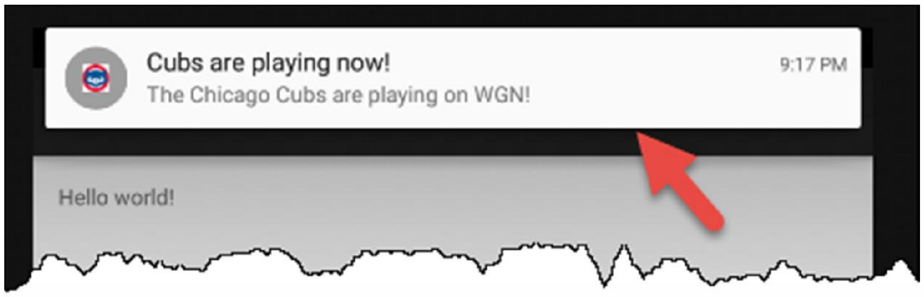
NOTIFICATION METADATA

New metadata – additional attributes – can be associated to notifications in Android 5. Notifications can have a category, a priority, and people (contacts) to a notification. These metadata can be used by the device to provide guidance (Android calls them "hints") about how to organize and display notifications.

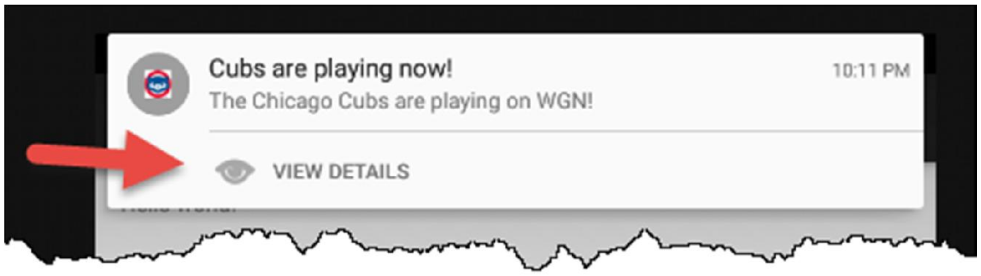
- category: the design documentation says all notifications should include a category and there are several category constants defined on the Notification class. They are just Strings and include categories such as "alarm", "promo" (for advertisements), "progress", etc. A category allows groups of notifications to be ranked and filtered – per user or system settings. For example, alarm notifications should display before promo notifications.
- priority: the priority can be set to min, max, high, low, or default – all again defined as constant ints on the Notification class. High priority notifications are displayed as heads-up notifications (discussed below) when accompanied by sound or vibration. Priority should be set with care. Max and high priority notifications risk interrupting the user in their current activity. Android provides a number of guidelines about setting the proper priority for your notifications (see here).
- person: adding a person to your notification allows the notification to be associated to a contact from the device's contacts. Adding a contact is accomplished through .addPerson(uri) method on the notification builder, where the URI is a person URI (it will even attempt to resolve mailto: and tel: schema URIs). Adding a person to the notification allows the device to group notifications from the same person and even prioritize/rank notifications from preferred persons.

HEADS-UP NOTIFICATIONS

Also new in Android 5 are Heads-up notifications. When the screen is unlocked and on, heads-up notifications appear like floating dialog boxes at the top of the smartphone over the status bar.


Cubs are playing now!
9:17 PM
The Chicago Cubs are playing on WGN!

When actions are associated with the notification, buttons display with the heads-up notification to allow the user to take immediate action based on the notification.



If not dismissed or acted on, the heads-up notifications fade away and return to the "regular" notifications in the status bar. Again, use heads-up notifications with care. These higher priority notifications that include sound and vibration are usually going to interrupt the device user's current work flow. You should have something of importance that needs immediate attention to use a heads-up notification.

In order to programmatically create a heads-up notification, the notification must be set with high priority (or better) and use either sound or vibration (the latter requiring a user permission in the manifest). Below is sample code to create a heads-up notification that vibrates and takes the user to a designated URL in a browser when the action is clicked.

```
Intent notificationIntent = new Intent(Intent.ACTION_VIEW);
notificationIntent.setData(Uri.parse("http://www.wgn.com"));
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);

Notification notification = new NotificationCompat.Builder(this)
 .setCategory(Notification.CATEGORY_PROMO)
 .setContentTitle(title)
 .setContentText(text)
 .setSmallIcon(icon)
 .setAutoCancel(true)
 .setVisibility(visibility)
 .addAction(android.R.drawable.ic_menu_view, "View details", contentIntent)
 .setContentIntent(contentIntent)
 .setPriority(Notification.PRIORITY_HIGH)
 .setVibrate(new long[]{1000, 1000, 1000, 1000, 1000}).build();
NotificationManager notificationManager =
 (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
notificationManager.notify(notification_id, notification);
```

DESIGN GUIDELINES

Before creating notifications, take a look at the design guidelines – particularly around iconography, category and priority.

Lollipop CardView

Android 5 has now been out for almost 2 months. Devices with Android 5 are starting to appear. No doubt, if you are an Android developer you are studying the new features and API. Take a look at one of my previous posts in Intertech's blog site, getAppTasks(), an earlier Android development tutorial.

CARDVIEW CONCEPT

In this Android development tutorial, we take a look at the new CardView. CardView was another widget type added in Lollipop. CardViews display "cards." Think about a collection of 3×5 cards you may use to collect recipes on. On the 3×5 card, you have some notes written out about the recipe. You may have written a link to the web site where you found the recipe on the card. You may also paste a picture of the final product.

Like a recipe card, the concept behind a card in the CardView is "a piece of paper that contains unique related data; for example, a photo, text, and link all about a single subject." Cards display content of different types. In particular, they are built to display a collection of objects whose size and associated actions vary.

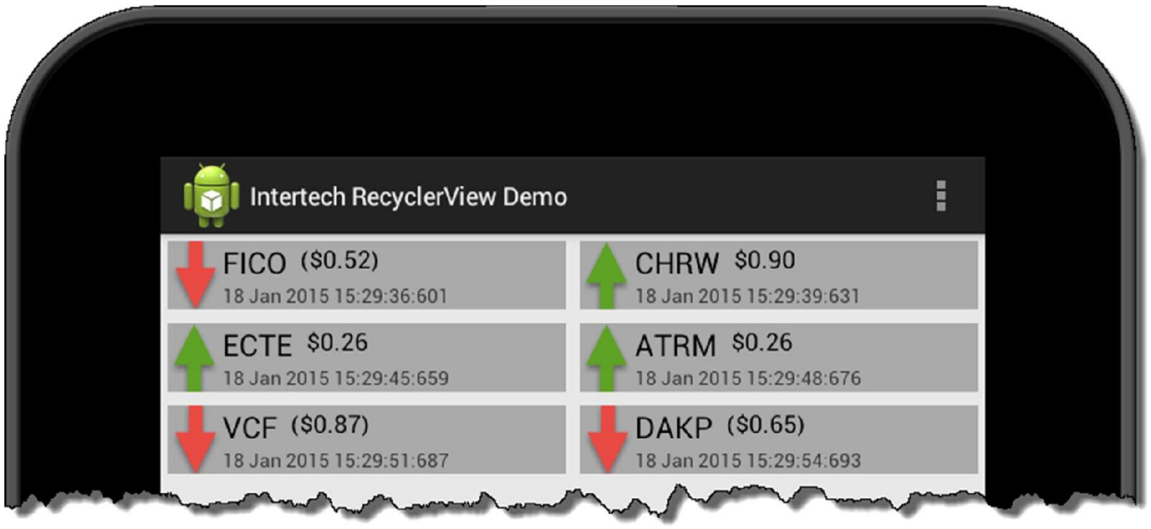
ANDROID CARDVIEW

Cards represented by CardViews can be used to display the row of data in a RecyclerView or ListView. Like the rows in a RecyclerView or ListView, a CardView typically serves as the entry to more detailed information – in a master/detail way. CardViews have a constant width but a variable height. The height is determined by the collection of objects displayed in the card.

In Android, CardView extends FrameLayout. Per the new Material Design, the CardView provides for the display of the card information in a panel with rounded corners, raised elevation and make themselves available to swipe gestures to move them.

CARDVIEW TUTORIAL

In my RecyclerView tutorial, I built an application that simulates a stock price ticker app – showing the rise and fall of a collection of financial stocks of interest. The stock prices were displayed in a RecyclerView – using a row layout to display the information about the stock price change. In this tutorial, we'll use a CardView to display each stock price change.



FICO ($0.52)
18 Jan 2015 15:29:36:601

CHRW $0.90
18 Jan 2015 15:29:39:631

ECTE $0.26
18 Jan 2015 15:29:45:659

ATRM $0.26
18 Jan 2015 15:29:48:676

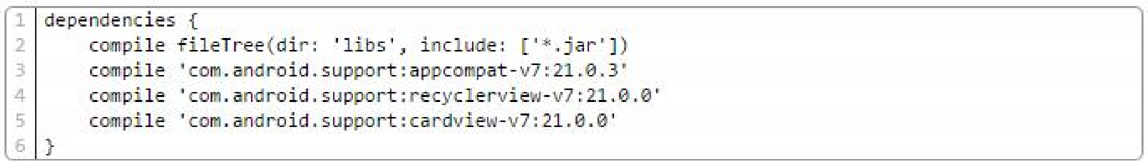
VCF ($0.87)
18 Jan 2015 15:29:51:687

DAKP ($0.65)
18 Jan 2015 15:29:54:693

To begin this Android development tutorial, download the RecyclerView tutorial code. The rest of this tutorial replaces the RecyclerView's rows with CardViews.

ADD THE CARDVIEW LIBRARY TO GRADLE

CardView is provided via a new Lollipop support library. Your first step in using CardView is to add the library to the dependencies list of your Gradle build file as shown by the last line in the code below (add it to the app's build.gradle and not the project build.gradle file).



```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile 'com.android.support:recyclerview-v7:21.0.0'
    compile 'com.android.support:cardview-v7:21.0.0'
}
```

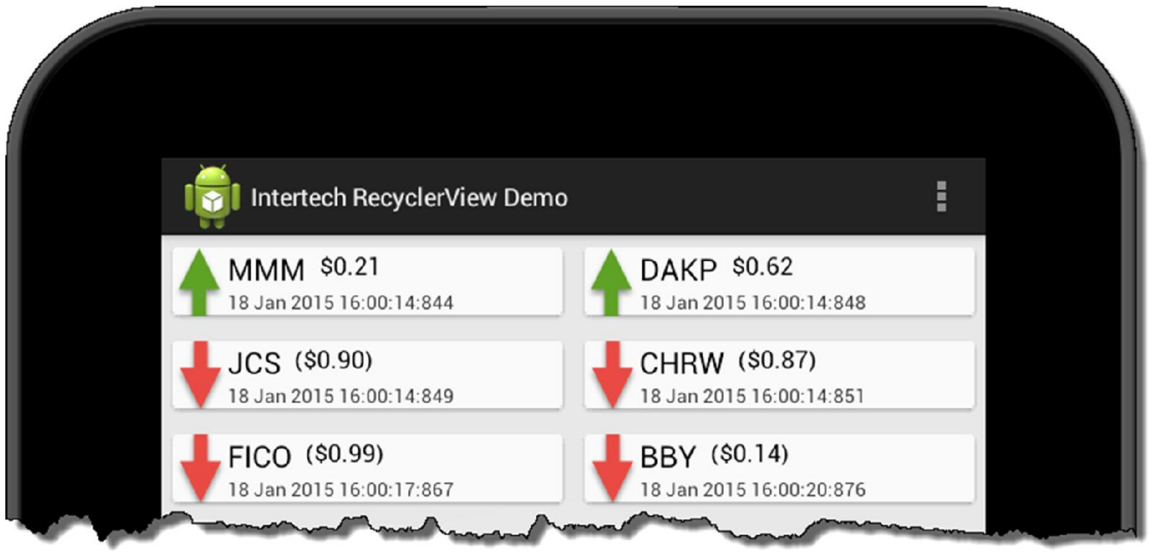
ADD CARDVIEW TO THE QUOTE LAYOUT

In the stock quote demo, you'll find a layout file called quote_item.xml. This layout defines the display of a single stock quote in the RecyclerView. Here we add CardView to put the information/contents about a new stock quote in a "card." Doing so is actually quite easy (and this should give you and idea of just how easy it is to add the card paradigm to your application).

Surround the RelativeLayout of the current quote_item.xml with a CardView. You will also want to remove the namespace and margin references in the RelativeLayout. The new CardView layout for quote_item.xml is shown below.

(see next page)

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="5dp"
    android:focusable="true"
    card_view:cardCornerRadius="4dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <ImageView
            android:id="@+id/directionImageView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:src="@drawable/down" />

        <TextView
            android:id="@+id/symbolTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_toEndOf="@+id/directionImageView"
            android:layout_toRightOf="@+id/directionImageView"
            android:text="symbol here"
            android:textAppearance="?android:attr/textAppearanceLarge" />

        <TextView
            android:id="@+id/priceTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_marginLeft="10sp"
            android:layout_toRightOf="@id/symbolTextView"
            android:text="price here"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/timestampTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/symbolTextView"
            android:layout_toEndOf="@+id/directionImageView"
            android:layout_toRightOf="@+id/directionImageView"
            android:text="timestamp here"
            android:textAppearance="?android:attr/textAppearanceSmall" />

    </RelativeLayout>
</android.support.v7.widget.CardView>
```

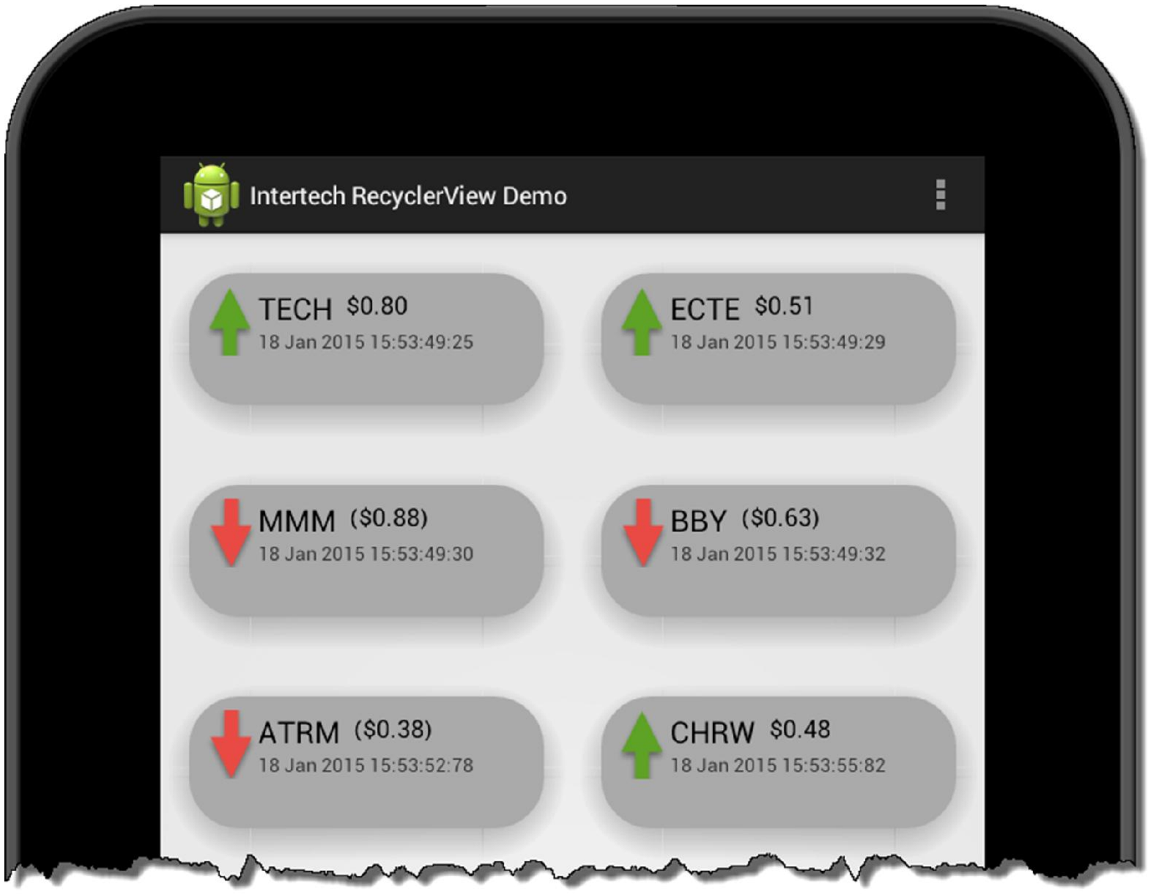


CARDVIEW ATTRIBUTES

At this point, there isn't much difference in the display. CardView allows for more Material Design adaptations. To demonstrate, let's change some of the CardView specific attributes in the layout. Below, I've added cardBackgroundColor, cardElevation, and cardCornerRadius attributes (some rather exaggerated to highlight some of the new design ideas).

```
<android.support.v7.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="5dp"
    android:focusable="true"
    card_view:cardBackgroundColor="@android:color/darker_gray"
    card_view:cardElevation="15dp"
    card_view:cardCornerRadius="36dp">
```

And now, the display has been altered significantly.



Of course, the CardView attributes can also be accomplished programmatically (some are easier to set than other). Remember that CardView is a FrameLayout wrapping each of the stock quotes in the RecyclerView. So in this case, adjust each of the CardView in the RecyclerView in the ViewHolder.

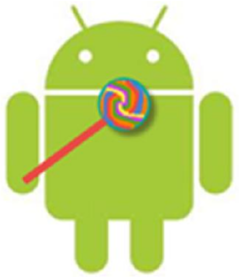
```
public QuoteViewHolder(View itemView) {
  super(itemView);
  directionImageView = (ImageView) itemView.findViewById(R.id.directionImageView);
  symbolTextView = (TextView) itemView.findViewById(R.id.symbolTextView);
  priceTextView = (TextView) itemView.findViewById(R.id.priceTextView);
  timestampTextView = (TextView) itemView.findViewById(R.id.timestampTextView);
  cardView = (CardView) itemView.findViewById(R.id.card_view);
  cardView.setCardElevation(15f);
  cardView.setRadius(36f);
}
```

IMPROVING THE UI

With RecyclerView and CardView in place, you can start to add additional gesture features to dismiss cards from the display. Check out this Github project to add swipe gestures to remove enclosed CardViews from the RecyclerView.

https://github.com/krossovochkin/Android-SwipeToDismiss-RecyclerView

Lollipop Material Design



One of the most publicized features of the Android 5 (Lollipop) release has been Material Design. Yet, it is probably one of the hardest features to grasp for those hearing about the feature for the first time. Google even set up a web site to try to describe and explain Material Design. Google has called it a “design language” or “visual language” for app development. Perhaps that term speaks to designers and marketing types. As a developer, when I first heard these terms, I was left wanting to understand exactly how it was going to impact my work in developing applications and how it was going to impact the look and feel of products I produced. I knew Material Design had something to do with the visual makeup, but how? In this post, I hope to take a developer’s look at Material Design – at least some of it. That is, I hope to explain Material Design by showing you its impact on app development.

As Wired reported, Google hopes to unite its product line under the design styles and principles established by Material Design. Given Google’s investment in Material Design and use beyond Android, we can probably expect Material Design to guide the look and feel of Android apps for the foreseeable future. That’s Android developer speak for – get used to it because your going to have to use it for a while. After reading this post, I highly encourage you to read the design specification and look at how design guidelines for Android have changed due to Material Design. For Material Design to succeed (and for Android devices to rival its competitor’s visual appeal), it is my opinion that we developers must follow its principals.

COMPONENTS OF MATERIAL DESIGN

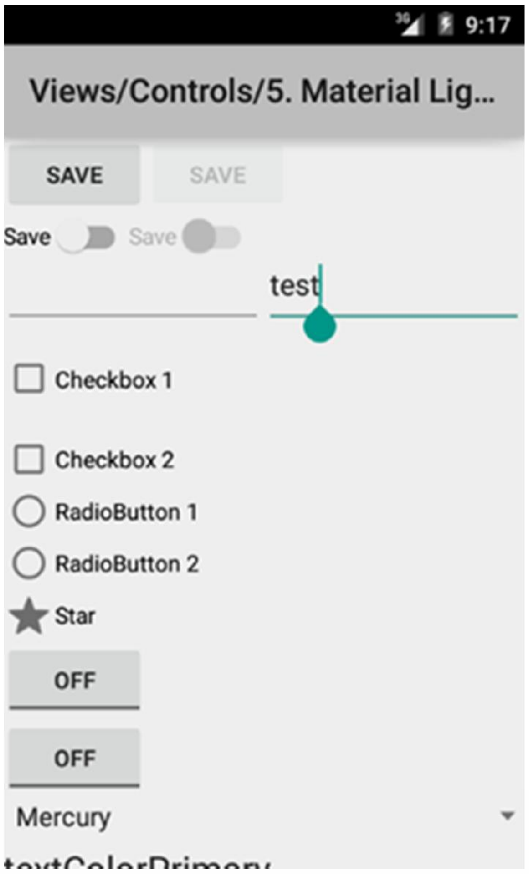
There are actually several parts to the Material Design.

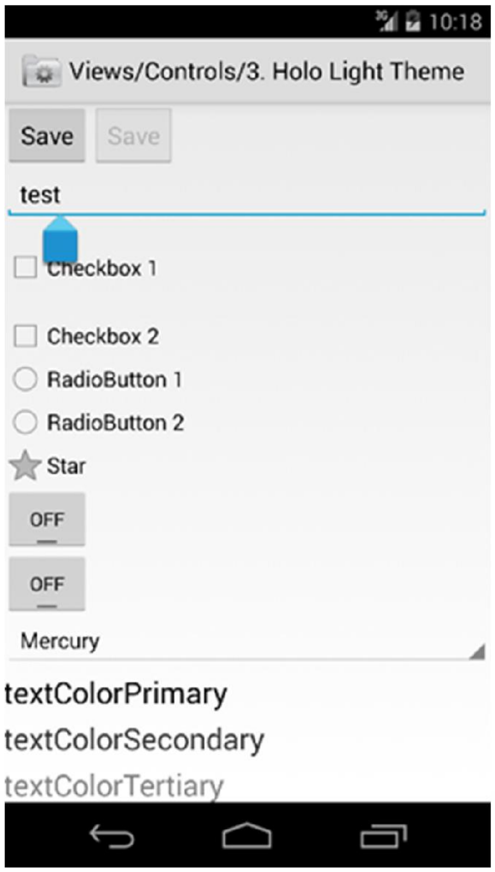
- A new material theme
- New widgets – particularly for displaying collections of data
- APIs for creating view elevations and shadows

- APIs for clipping views
- Addition of vector drawables
- APIs for custom animations

MATERIAL THEME

A style in Android is a collection of properties that specify the look and format for a View. A theme is a style applied to an entire Activity or application. If you have developed in Android for even a little while, you are probably familiar with new themes (and thus styles) that have been created with most of the major Android releases. Android 5 is no different in this respect. There are new themes (Material Light Theme, Material Dark Theme and Material Light with Dark Action Bar Theme to be precise) that have been released with Lollipop. The API Demo application provided with the AVDs can give you insight into these new themes – especially as they differ from the Holo themes (which debuted with Android 4). Below, on the left is the API Demo app (Views -> Controls->Material Light) run on an Android Lollipop AVD displaying the Material Light theme. On the right is the API Demo app (Views->Controls->Holo Light) run on an Android KitKat (4.4) AVD.





Importantly, the Material Design themes allow for activity transition animations, touch feedback animations and a color palette that allows for easier branding of the app. The color palette can even be set by the colors found in a drawable (more on this later). The color palette allows the colors of the status bar to be easily customized, which design guidelines are now suggesting you do in Android 5.

"The status bar should almost always have a clear delineation from the primary toolbar"

NEW WIDGETS

As reported in early posts in this blog, Android 5 also introduces now widgets to simplify and improve the display of collections of data. The RecyclerView scrolls more efficiently especially when considering large data sets and data sets whose elements are changing dynamically. The RecyclerView also offers some nice visual additions like item animation that is representative of the new Material Design.

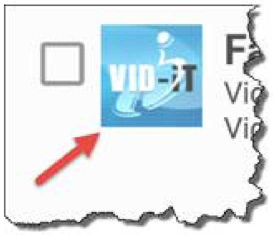
CardViews can be used to display the row of data in a RecyclerView or ListView. In Android 5, CardView extend FrameLayout. Per the new Material Design, the CardView provides for the display of the card information in a panel with rounded corners, raised elevation and make themselves available to swipe gestures to move them.

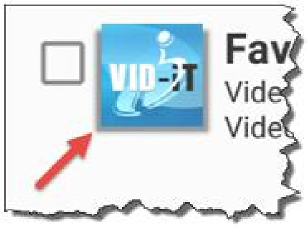
VIEW ELEVATION AND SHADOWS

Widgets (more formally Views) always had an X and Y position on the screen. Now they have a Z position. The Z position defines the virtual elevation of the widget from the screen surface in order to give it the appearance as if it is hovering – and thus projecting a shadow of the view to the surface below. The higher the Z value of the view, the larger the shadow cast by the widget. Further, views with a higher Z value "occlude" (I had to look it up myself if you are not sure what it means – it means to obstruct) views with smaller Z values as if they were to hover above the images with smaller Z values.

To see the new shadow/Z value projected, simply set the elevation property to a particular density pixel (or other spatial unit of measure) value. Here is an example of an ImageView with no elevation (on the left) versus an elevation set (as shown in the code below) to 15dp (right).

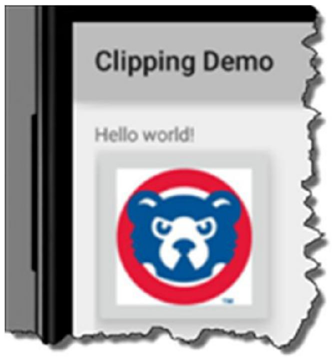
```xml
<ImageView
  android:id="@+id/favoriteItemThumbnailIV"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentTop="true"
  android:layout_toEndOf="@+id/favoriteItemCB"
  android:layout_toRightOf="@+id/favoriteItemCB"
  android:elevation="15dp"
  android:src="@drawable/vidit" />
```





CLIPPING VIEWS

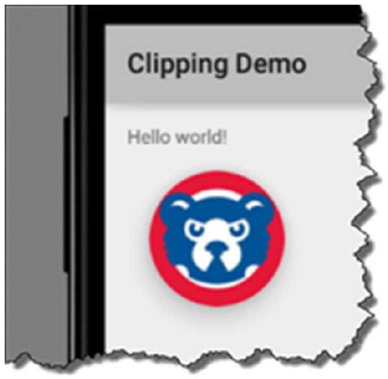
In previous versions of Android, soft corners, round buttons, etc. all had to be faked by using the appropriate color shading in images, backgrounds, etc. Android 5 now supports clipping to give views the appropriate shape. Specifically, you need to create and apply a ViewOutlineProvider (new in Android 5) to your view to have the view clipped. Suppose you have a standard ImageButton with a PNG drawable set as its source. Below, the Cubs baseball team logo serves as the source PNG to the ImageButton.



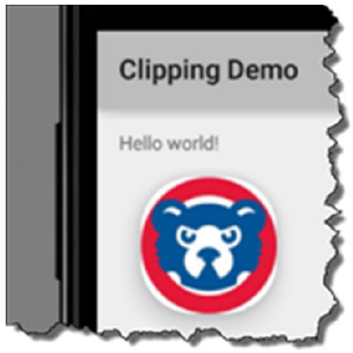
As you can see, the actual logo is circular in shape. Wouldn't it be nice if the button could be the shape of the logo – that is circular? In Android 5 it can, just define a ViewOutlineProvider to clip the view appropriately – in this case, clip the view according to the dimensions of the logo in a circular / oval fashion. Here is how it is done in code.

```java
final ImageButton cubsButton = (ImageButton) getActivity().findViewById(R.id.cubsButton);
final int paddingSize = getResources().getDimensionPixelSize(R.dimen.logopadding);
final int imageSize = cubsButton.getDrawable().getIntrinsicWidth();
ViewOutlineProvider provider = new ViewOutlineProvider() {
  @Override
  public void getOutline(View view, Outline outline) {
    outline.setOval(paddingSize,paddingSize,imageSize+paddingSize, imageSize+paddingSize);
  }
};
cubsButton.setOutlineProvider(provider);
cubsButton.setClipToOutline(true);
```

The results of the clipping are shown below – a “round” ImageButton.



A similar effect can be had by using a rounded rectangle as the means to provide the clipping. Changing the outline.setOval(…) line of code above to outline.setRoundRect(paddingSize,paddingSize,imageSize+paddingSize, imageSize+paddingSize, (imageSize+paddingSize)/3) yields a similar yet slightly different clipped view.

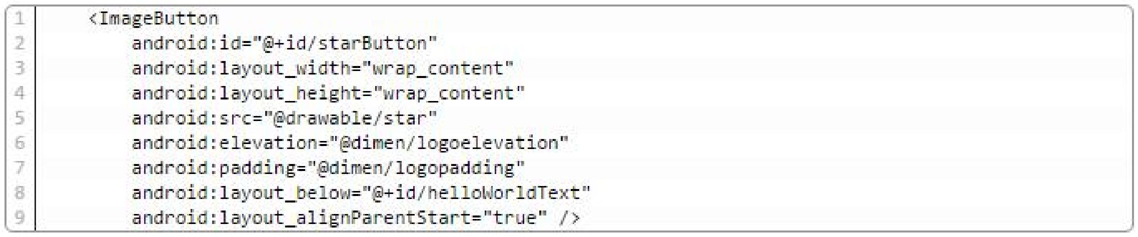


SVG DRAWABLES

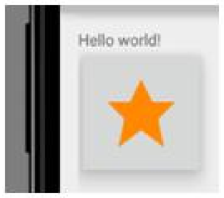
Scalar vector graphics (SVG) are now supported in Android 5. In prior releases, images – or drawables as they are know – were in the form of png, jpg, or gif images. Create a resource which defines the graphic using the SVG Path API. Below, I have defined an orange star using SVG Path in a file I called star.xml in the res/drawables folder.

```xml
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="80dp"
    android:height="80dp"
    android:viewportHeight="80"
    android:viewportWidth="80">

    <path
        android:fillColor="@android:color/holo_orange_dark"
        android:pathData="M10 30 L30 30 L40 10 L50 30 L70 30 L55 45 L60 70 L40 55 L20 70 L25 45 Z"
</vector>
```

The vector file can then be applied as if it was any other image. Take, for example, its use as the source image in an ImageButton. You even refer to the SVG image as a @drawable!



```
    <ImageButton
        android:id="@+id/starButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/star"
        android:elevation="@dimen/logoelevation"
        android:padding="@dimen/logopadding"
        android:layout_below="@+id/helloWorldText"
        android:layout_alignParentStart="true" />
```

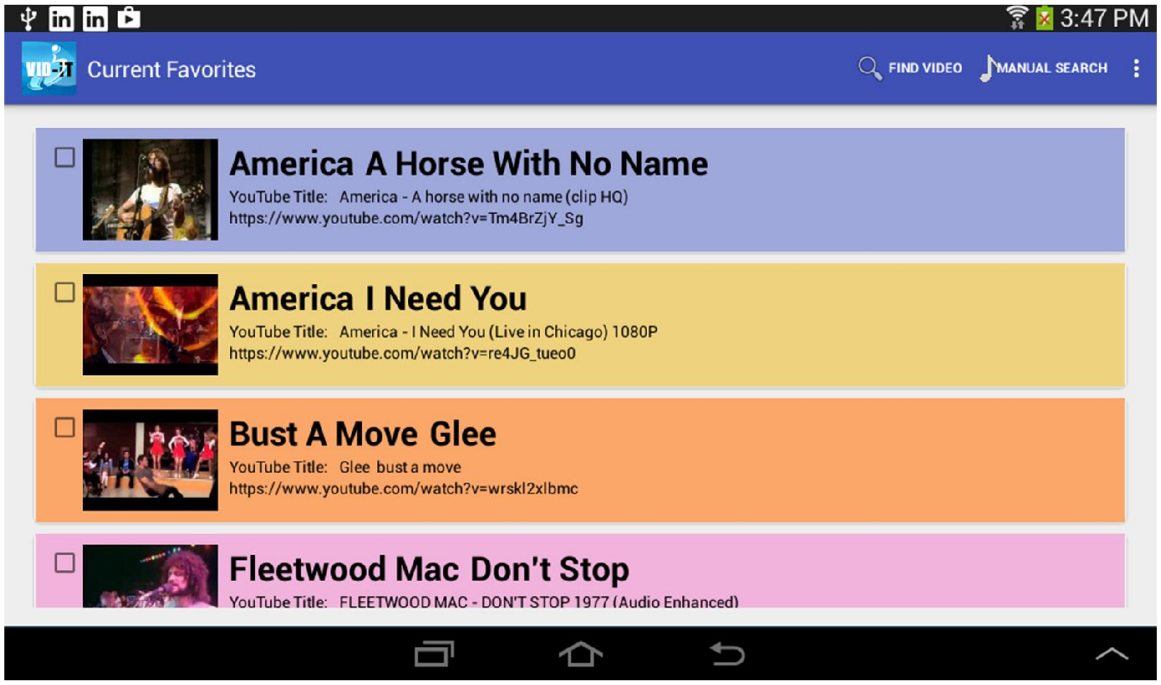


Lollipop comes with even more features around drawables. Drawables can now be defined with an alpha mask for tinting, and you can extract prominent colors from an image to help colorize the rest of your application components.

PALETTE & COLOR EXTRACTION

I used the color extraction API in the latest version of Vid-iT available at Google Play. Vid-iT allows people to find YouTube videos matching the music collection stored on their Android device. I used the color extraction API to extract the prominent colors of the video thumbnail image to color up the CardView used to display each video in Vid-iT. Note how each song/video listing below has a background color that seems to resemble the most prominent color in the video thumbnail?

(see next page)



America A Horse With No Name
YouTube Title: America - A horse with no name (clip HQ)
https://www.youtube.com/watch?v=Tm4BrZjY_Sg

America I Need You
YouTube Title: America - I Need You (Live in Chicago) 1080P
https://www.youtube.com/watch?v=re4JG_tueo0

Bust A Move Glee
YouTube Title: Glee bust a move
https://www.youtube.com/watch?v=wrskl2xlbmc

Fleetwood Mac Don't Stop
YouTube Title: FLEETWOOD MAC - DON'T STOP 1977 (Audio Enhanced)

The API that makes this possible is the new Pallete API found in Lollipop. A Palette is a helper class used to extract colors from an image (as shown below).

```
Bitmap bitmap = ((BitmapDrawable) imageView.getDrawable()).getBitmap();
Palette palette = PaletteTransformation.getPalette(bitmap);
layout.setBackgroundColor(palette.getLightVibrantColor(0));
```

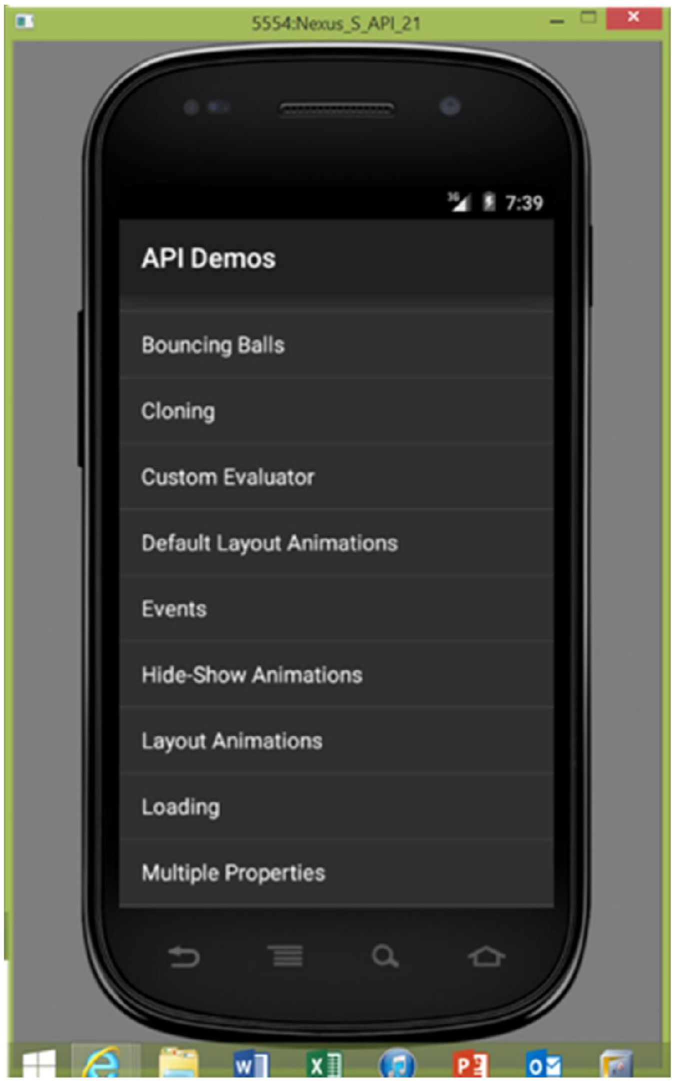
Depending on the colors of the image, Palette attempts to define the prominent dark and light contrasting colors from the image. Here is the list of color definitions it tries to come up with.

- Vibrant
- Vibrant Dark
- Vibrant Light
- Muted
- Muted Dark
- Muted Light

Above, in the Vid-iT example code, the light vibrant color is used to set the CardView for each favorite video displayed. Palette can be a very helpful tool in branding your application in a way that is consistent with a company logo, or color scheme used by an organization. White label applications can now be easily "colorized" based on the user or user's organization with Palette.

ANIMATIONS

The addition of animations has been added to several areas of the user interface in Lollipop. In the tutorial on RecyclerView, I showed how animation can easily be added to highlight the addition/removal of items to the RecyclerView. Additionally, the animation API allows developers to highlight activity transition and show state changes in views (like button clicks). The API Demo application, provided with Android 5 AVDs, has a collection of animation samples to show you just how extensive this new API is. Try them out to get inspired by some of these new features. Let's face it, animation had been one of user interface areas where iOS was superior to Android. Lollipop's Material Design animation API is helping to close that gap.



Job Scheduler

Project Volta is the name given to features of Android 5 aimed at improving power usage on Android devices. Volta includes better tools for monitoring battery usage. I'll be covering those tools in the next tutorial. It also includes a new API for scheduling work – a job scheduler – which I cover in this Android Development Tutorial.

You might ask why is a job scheduler API part of a power improvement effort? The idea is that often times, the battery life of our devices is better served when more taxing tasks are performed under certain conditions – like when the device is plugged in or connected via WIFI versus using the mobile carrier network. Of course, the job scheduling API can also be used to schedule asynchronous work that isn't always about battery improvement. Sometimes there are tasks that just need to occur regularly or when the device is otherwise unoccupied.

Now scheduling work in Android is not new. The AlarmManager in Android, for example, has been a means to repetitively schedule an application or work to kick off since Android API 1. The new job scheduling API differs from the AlarmManager in that it is more aware of the environmental resources and can kick of batch jobs when certain resources are more available.

Job Scheduling API

There are several new classes in Android 5 to help create and execute your scheduled tasks.

JOBSCHEDULER SERVICE

There is a service for most things in Android. A new service, the JobScheduler Service, helps you manage scheduled tasks. Obtain the JobScheduler service via getSystemServices( ).

JobScheduler jobScheduler = (JobScheduler)
getApplicationContext().getSystemService(JOB_SCHEDULER_SERVICE);

Call getSystemServices( ) on a Context – the global application context is used in the example above.

The scheduler has methods to schedule, cancel and see the list of pending jobs. Jobs are represented by JobInfo objects.

JOB SCHEDULING API

You now need to build a job and schedule it for execution. Again a familiar pattern is used in Android to help you create a scheduled job – the builder. Use a Builder to create a new JobInfo object.

```
ComponentName componentName = new ComponentName(getApplicationContext(),TestService.class);
JobInfo jobInfo =  new JobInfo.Builder(1,componentName).setMinimumLatency(1000).build();
```

The Builder requires the JobService or "job" name (the name is TestService in the example code above) and the Job ID. The Job ID (1 in the example above) can be used to cancel the job via the JobScheduler.

Note the call to setMinimumLatency( ) before the call to build the JobInfo object? This is the first of many optional method calls used when creating the JobInfo.

The JobInfo object encapsulates all the information about the work you want accomplished and the conditions of its execution (time, power availability, etc.). JobInfo isn't the actual work or task itself. The actual work to be accomplished is defined in a JobService (like TestService above). The JobInfo just represents the parameters or criteria about when you want the work to execute. It is in the JobInfo object, for example, that you specify that a big downloading job is to run when the device is plugged in and has a WIFI connection.

Optional parameters or criteria to the JobInfo include:

- make the job periodic (a repeating job within a given period)
- the interval between periodic job runs
- the maximum execution delay – how long to wait to kick of the job (not a parameter for a periodic job task)
- the minimum amount of time delay before starting the job (not a parameter for a periodic job task)

- the device must be plugged in and charging in order to run the job
- the device must be idle ("in an idle maintenance window") in order to run the job
- the device must have a network connection (either WIFI, unmetered, or any)
- the device must not be connected to a network
- the job should be rescheduled if the device reboots (called a "persistent" job).

* all time related parameters are specified in milliseconds.

Lastly, you can also set what is called a back off policy with the JobInfo. The backoff policy applies to tasks that have finished and need to be retried (perhaps because of a failure or the need to do the task again). The backoff policy allows the job to be retried, but under additional parameters like a delay before rescheduling the job or even growing the delay in an exponential way for repeated failures. Consider the situation whereby your backend service goes down for some extended period. During that time, the mobile application running on multiple devices may all want to keep retrying until the service comes back up. The idea behind the backoff policy is that is allows you to determine how to stagger and/or ease retries so that once the backend server comes back on line, there is not a glut of request from the mobile clients – possibly causing it to go down again.

Each of the optional parameters are set with a method call to the builder. Here, for example, is how to specify that you want your job (again, defined by TaskService) to kick off only when there is any network connection, when the device is plugged in, and repeats at least every 12 hours (43200000 milliseconds).

```
ComponentName componentName = new ComponentName(getApplicationContext(), TestService.class);
JobInfo jobInfo = new JobInfo.Builder(1, componentName).setPeriodic(43200000)
  .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY).setRequiresCharging(true).build();
```

CREATING THE SERVICE

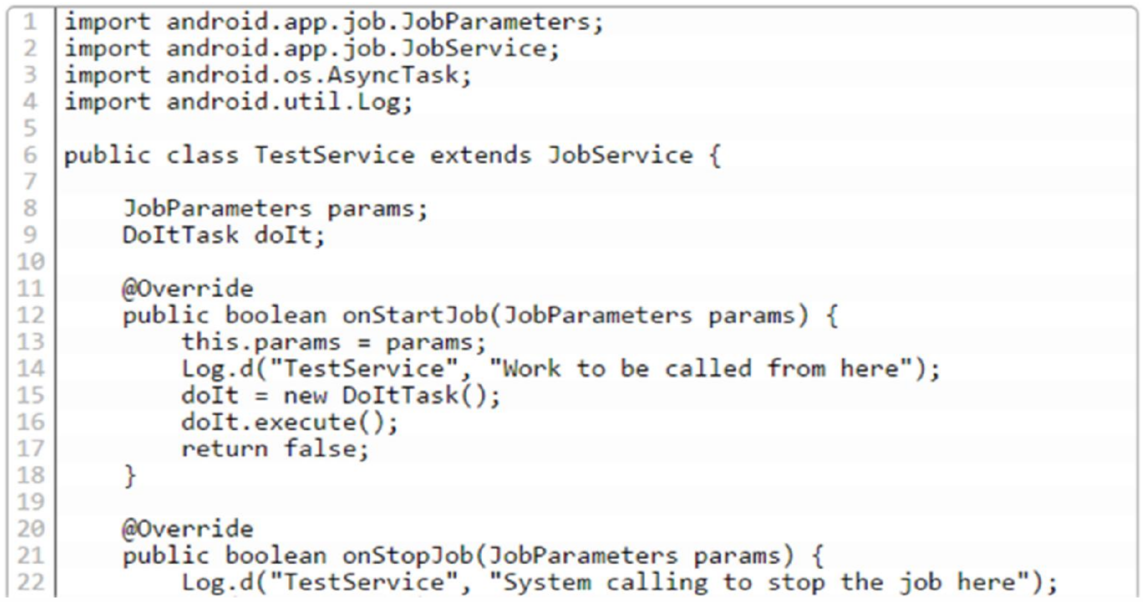
Where is the work of the job specified? The object that contains the job or work to be accomplished is an instance of JobService. JobService (android.app.job) is a subclass of the the good-old Android Service class. As such, it represents background (non-UI) work that is to accomplished. It is important to note that by default, just as with regular Android Service objects, the JobService runs on the main / UI thread. As with a regular Service, you must take care of getting the execution of the task to a non-UI thread – such as creating and using an AsyncTask. (refer to above tutorial about thread options and how to deal with other threads and communication with the UI thread).

Unlike the super class Service, you do not override the onStartCommand( ) method in a JobService. Instead, the JobService has new methods: onStartJob( ) and onStopJob( ). The onStartJob( ) method gets called when the JobScheduler runs your job per the conditions specified in the parameters of your JobInfo. The onStartJob( ) method runs on the main / UI thread. It is typically in this method that you create and kick off another thread to take care of the work of your job. The onStartJob( ) method returns a Boolean true if the service needs to process the work and false if there is no more work to be done for the job.

The onStopJob( ) method gets called by the system when your job is executing and it detects that the criteria or parameters associated with your job via the JobInfo no longer apply and it must stop your job. For example, if you specified in your JobInfo that the job must have network access to run, the onStopJob( ) will get called when network access is no longer available. This method also executes on the main / UI thread. The onStopJob( ) method also returns a Boolean to indicate whether you would like the job to rescheduled / retried when conditions are right again.

One other very important method built in to the JobService superclass is a method that you cannot override. The final jobFinished( ) method must be called by your code when your job is done executing. This notifies the system’s “JobManager” that it is finished so that it no longer needs to be managed, rescheduled, etc. One of the parameters (a Boolean true) to jobFinished( ) allows you to indicate that you want the job to be rescheduled according to the back-off criteria.

Here is a complete example of the example JobService – TestService – with its onStartJob( ) and onStopJob( ) methods along with an inner AsyncTask class used to demonstrate how to possibly move the actual work to a separate thread.



```java
import android.app.job.JobParameters;
import android.app.job.JobService;
import android.os.AsyncTask;
import android.util.Log;

public class TestService extends JobService {

    JobParameters params;
    DoItTask doIt;

    @Override
    public boolean onStartJob(JobParameters params) {
        this.params = params;
        Log.d("TestService", "Work to be called from here");
        doIt = new DoItTask();
        doIt.execute();
        return false;
    }

    @Override
    public boolean onStopJob(JobParameters params) {
        Log.d("TestService", "System calling to stop the job here");
        if (doIt != null)
            doIt.cancel(true);
        return false;
    }

    private class DoItTask extends AsyncTask<Void, Void, Void> {
        @Override
        protected void onPostExecute(Void aVoid) {
            Log.d("DoItTask", "Clean up the task here and call
jobFinished...");
            jobFinished(params, false);
            super.onPostExecute(aVoid);
        }

        @Override
        protected Void doInBackground(Void... params) {
            Log.d("DoItTask", "Working here...");
            return null;
        }
    }

}
```

Now, all that is left is to show you how to start the JobService, given the JobInfo and JobScheduler. Note the last line in the code below.

```
JobScheduler jobScheduler = (JobScheduler)
getApplicationContext().getSystemService(JOB_SCHEDULER_SERVICE);
ComponentName componentName = new ComponentName(getApplicationContext(),
TestService.class);
JobInfo jobInfo = new JobInfo.Builder(1,
componentName).setOverrideDeadline(10).setRequiresCharging(true).build();
jobScheduler.schedule(jobInfo);
```

WRAP UP

App power utilization is serious business. At last year's AnDevCon conference, several speakers indicated that one of the best ways to get yourself a low rating on Google Play is to consume too much power. Users don't like their phones and tablets going dead because of your poor decision making. So use the JobScheduler to schedule power-expensive tasks when it is unlikely to affect the device's battery life.

Project Volta

PROJECT VOLTA

In the previous Android development tutorial on Android Lollipop JobScheduler, I mentioned Project Volta as part of a large, multi-faceted initiative associated with Android 5 (Lollipop) to improve the battery utilization and life on Android devices. In that last tutorial, I covered the JobScheduler API, which was part of Project Volta. The JobScheduler API provides the means to schedule taxing tasks under certain device conditions – like when the device is plugged in or connected via WiFi versus using the mobile carrier network – thereby improving battery life by simply doing less work when the battery is needed.

Google discovered that just waking the device from sleep for a single second required two minutes of standby time. The "wake-up" calls turn on the screen, the processors and radios to get incoming data. That's a lot of energy consumed for an action that can be delayed (say when it is being charged) or simply ignored (say when the device is in airplane mode) when appropriate guidance is provided to a task that needs to be accomplished.

HOW MUCH BATTERY SAVINGS?

It is/was Google's hope that Android 5 would give users an average of 90 minutes more of battery life a day. Early indications about the impact of Project Volta and Lollipop on Android battery life are mixed. Some early reports suggest that the newest Android version is giving devices as much as 36% more battery life (see here). Other reports (like this one) are suggesting that there isn't any significant savings yet, but that the true impact may not be felt yet since most apps are not using the new features provided by Volta and Android 5. Certainly, it would appear that the device and usage greatly influence the battery life – just as they do today.

LOLLIPOP/PROJECT VOLTA SPECIFICS

What changes and additions were made to improve the battery life of Android – beyond the JobScheduler API already covered?

Battery Saver Mode

A new battery saver mode was added to the OS. Device users can turn this feature of the device on/off (it is off by default) – more on how in a second.

What the battery saver mode does is reduce background processing (like fetching new emails frequently), throttle down the device's performance, eliminate unneeded screen animations, turn off auto application updates, and even dim the screen a little.

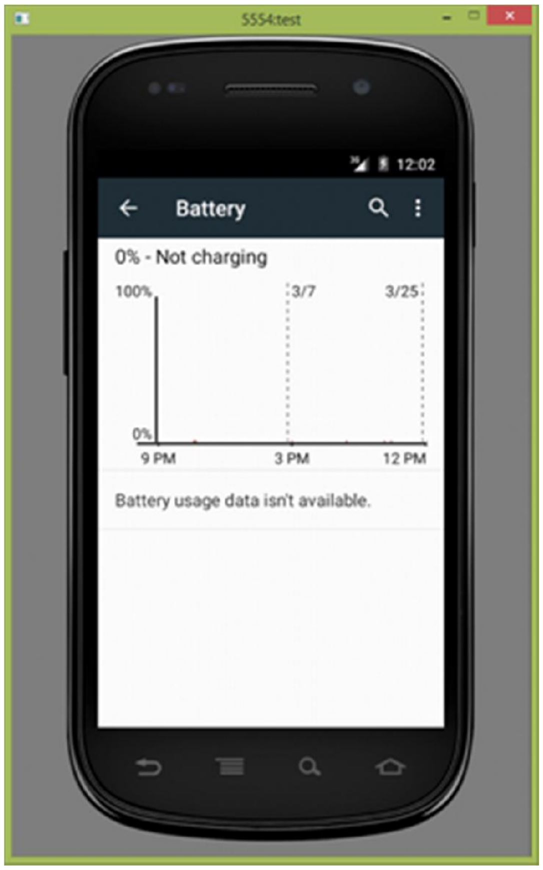
Things may seem a little more sluggish and not as "snazzy" as when in non-battery saver mode, but the idea is it will extend the life of the battery in exchange for these inconveniences. Battery saver mode is switched off when your device is reconnected to a power outlet.

As mentioned, battery saver mode is an opt-in feature in the new Android 5 devices. That means users most turn it on through settings in order to get the battery savings. It won't just happen by default! The easiest way to get to the new battery savings options in settings is to use the new search feature in Settings. Take a look at the small video below to see how to turn on the battery saver mode.

(click to see video)

Note too that you can set the battery saver to automatically come on when the device has 5% or 15% battery life left.

While on the Battery settings page, you might notice that your device (or the AVD as shown below) offers some battery usage statistics. These displays are getting more and more sophisticated and detailed in providing users insight into power-hungry apps. While not consistent across device platforms yet (in the display or usage), Android 5 is putting focus on power usage and you can expect users to start to use these types of stats to make more decisions about what apps stay and go on to their devices.



0% - Not charging

Battery usage data isn't available.

ART

The new Android Runtime (ART) was introduced as an optional replacement for the Dalvik Virtual Machine (DVM) with Android 4.4. Developers could choose to run their apps on ART versus DVM in 4.4. With Lollipop, ART is the default virtual machine – goodbye DVM.

ART uses an ahead-of-time (AOT) compiler to turn an app's bytecode to native executable code. This is done just one time when the app is first loaded and started. The old DVM used a just-in-time compiler which compiled the app's code each time the app was opened. The trade off is speed and battery drain (better in ART then in the DVM) versus space. The native code typically uses more space and now that must be stored by the device after it is first compiled. Unless you have a lot of apps/data, the larger footprint is not typically a big a deal given the larger storage spaces of today's devices.

Google has also optimized the code compiled by ART that again improves performance (fewer CPU cycles) and saves battery life as the app gets used.

Finally, a new and vastly improved garbage collection system is in place with ART. Again, a more improved GC means better performance and better battery life.

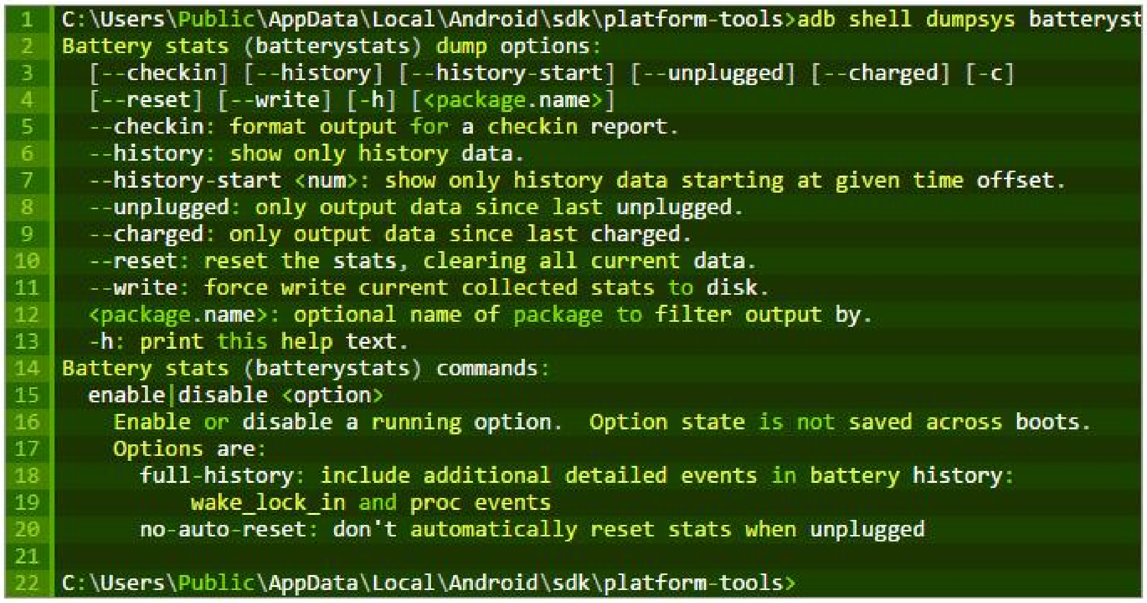
All these ART improvements come without the need for developers to write or deploy their applications any differently.

By the way, ART also supports 32 bit and 64 bit processors in the x86 and MIPS architectures. In the long run, ART provides a great deal of advantages to Android that go beyond power savings.

BATTERY STATS

The final improvement Volta brings to the Android developer is a new dumpsys tool option that provides battery statistics information. The dumpsys tool runs on a device or AVD (typically through the adb shell) and it provides statistical data about the services and operations on that device or AVD. Developers have been using this tool for years to get insight into what is happening on the device such finding out CPU utilization or the state/configuration of the Wifi. Find more details about the tool in general here.

A new option to dumpsys provides a deep collection of battery statistics. This option is called batterystats. You can find out more details about theses stats here. If you take a look at the documentation, the tool can provide a lot of detail and there are command options to filter away some of that detail. Importantly, there is a -h (for help) switch to get an appreciation of the batterystats options. Below, the adb shell command is used to invoke batterystats to get the help (the help results shown below the command call).

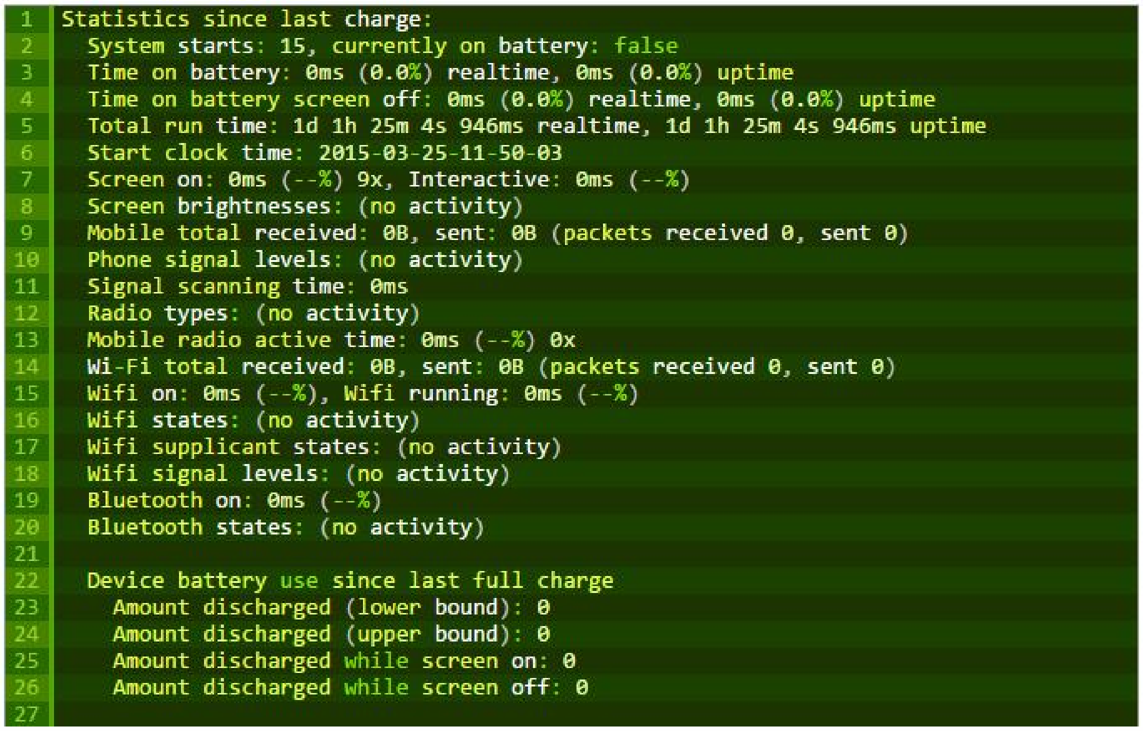


```
C:\Users\Public\AppData\Local\Android\sdk\platform-tools>adb shell dumpsys batteryst
Battery stats (batterystats) dump options:
  [--checkin] [--history] [--history-start] [--unplugged] [--charged] [-c]
  [--reset] [--write] [-h] [<package.name>]
  --checkin: format output for a checkin report.
  --history: show only history data.
  --history-start <num>: show only history data starting at given time offset.
  --unplugged: only output data since last unplugged.
  --charged: only output data since last charged.
  --reset: reset the stats, clearing all current data.
  --write: force write current collected stats to disk.
  <package.name>: optional name of package to filter output by.
  -h: print this help text.
Battery stats (batterystats) commands:
  enable|disable <option>
    Enable or disable a running option.  Option state is not saved across boots.
    Options are:
      full-history: include additional detailed events in battery history:
          wake_lock_in and proc events
      no-auto-reset: don't automatically reset stats when unplugged

C:\Users\Public\AppData\Local\Android\sdk\platform-tools>
```

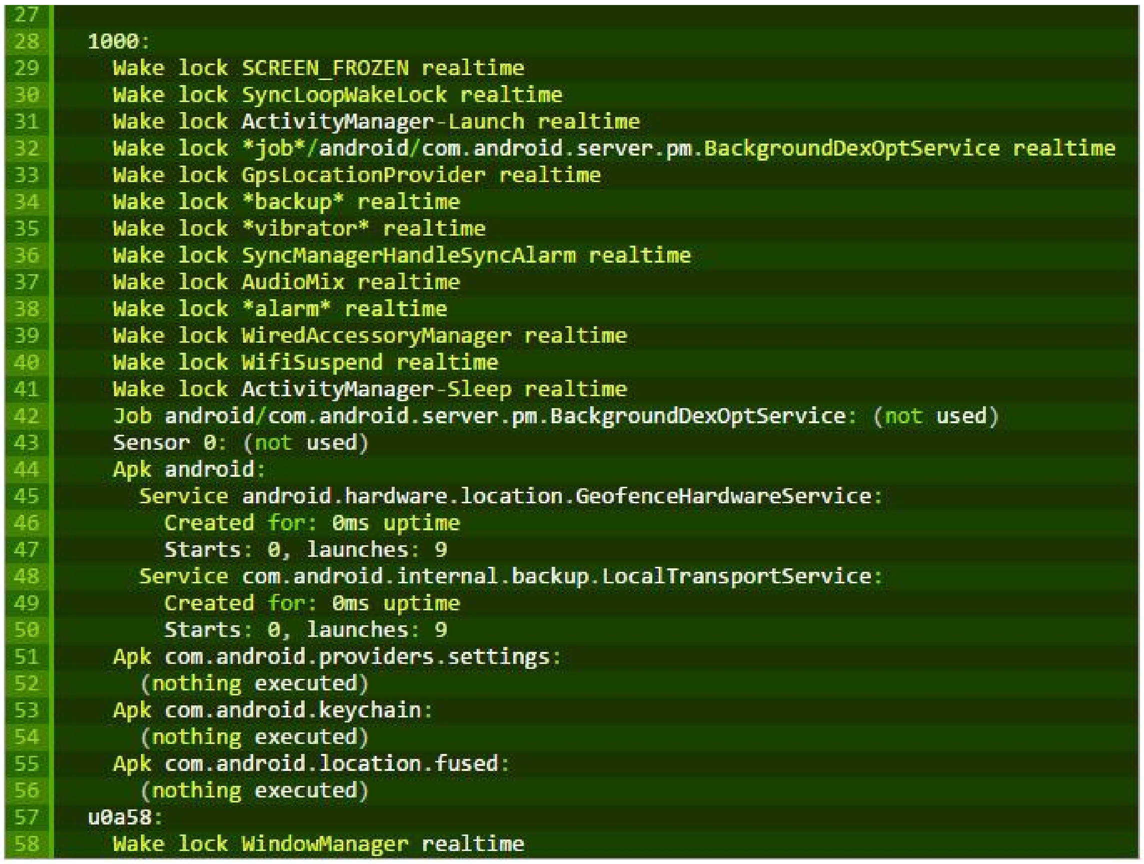
As a simple example of using dumpsys batterystats, the request below asks for the battery stats on the application package named com.intertech.vidit through the adb shell.

adb shell dumpsys batterystats com.intertech.vidit

Here are the results of such a request.



```
Statistics since last charge:
  System starts: 15, currently on battery: false
  Time on battery: 0ms (0.0%) realtime, 0ms (0.0%) uptime
  Time on battery screen off: 0ms (0.0%) realtime, 0ms (0.0%) uptime
  Total run time: 1d 1h 25m 4s 946ms realtime, 1d 1h 25m 4s 946ms uptime
  Start clock time: 2015-03-25-11-50-03
  Screen on: 0ms (--%) 9x, Interactive: 0ms (--%)
  Screen brightnesses: (no activity)
  Mobile total received: 0B, sent: 0B (packets received 0, sent 0)
  Phone signal levels: (no activity)
  Signal scanning time: 0ms
  Radio types: (no activity)
  Mobile radio active time: 0ms (--%) 0x
  Wi-Fi total received: 0B, sent: 0B (packets received 0, sent 0)
  Wifi on: 0ms (--%), Wifi running: 0ms (--%)
  Wifi states: (no activity)
  Wifi supplicant states: (no activity)
  Wifi signal levels: (no activity)
  Bluetooth on: 0ms (--%)
  Bluetooth states: (no activity)

  Device battery use since last full charge
    Amount discharged (lower bound): 0
    Amount discharged (upper bound): 0
    Amount discharged while screen on: 0
    Amount discharged while screen off: 0

```



```

  1000:
    Wake lock SCREEN_FROZEN realtime
    Wake lock SyncLoopWakeLock realtime
    Wake lock ActivityManager-Launch realtime
    Wake lock *job*/android/com.android.server.pm.BackgroundDexOptService realtime
    Wake lock GpsLocationProvider realtime
    Wake lock *backup* realtime
    Wake lock *vibrator* realtime
    Wake lock SyncManagerHandleSyncAlarm realtime
    Wake lock AudioMix realtime
    Wake lock *alarm* realtime
    Wake lock WiredAccessoryManager realtime
    Wake lock WifiSuspend realtime
    Wake lock ActivityManager-Sleep realtime
    Job android/com.android.server.pm.BackgroundDexOptService: (not used)
    Sensor 0: (not used)
    Apk android:
      Service android.hardware.location.GeofenceHardwareService:
        Created for: 0ms uptime
        Starts: 0, launches: 9
      Service com.android.internal.backup.LocalTransportService:
        Created for: 0ms uptime
        Starts: 0, launches: 9
    Apk com.android.providers.settings:
      (nothing executed)
    Apk com.android.keychain:
      (nothing executed)
    Apk com.android.location.fused:
      (nothing executed)
  u0a58:
    Wake lock WindowManager realtime
```

Note, earlier versions of Android provided a weaker dumpsys command to get battery statistical information – it was called batteryinfo. With Android 5, this option to dumpsys is considered deprecated (see here).

BATTERY HISTORIAN

Along with battery stats, a Google tool – called Battery Historian – is available to put the battery stats data in a visual display. You can find the tool and information about it in GitHub. Unfortunately, the tool is a Python-based script and requires you to set up Python. I am sure we can look for more tools and even IDEs that will provide us insight into the data provided by batterystats in the future.

WRAP UP

I hope you have found my collection of Android Development Tutorials on Android 5 to be helpful. Again, this is a major release with many more features that I have not covered. Both users and developers will see a lot of new capability that will take some time to get used to.

You can also download a copy of the presentation I made at DevFest MN about the new features of Android 5.

NEED FURTHER ASSISSTANCE?

See what Intertech has to offer with our Android Training and/or Consulting offerings. We will help you become a better developer by furthering your knowledge on Android Lollipop as well as the entire platform and help your organization utilize the Android platform in your mobile strategies. Click through below to learn more.

- Intertech's Android Training Offerings
- Intertech's Android Consulting Offerings