Android Programming: Developing Custom Components

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

© 2012 Marty Hall

For live Android training, please see courses at http://courses.coreservlets.com/.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Custom components: big idea**
- **Using onMeasure**
  - If told how big to be: resizing content to fit current size
  - If asked how big you want to be: calculating desired sizes
- **Reading custom XML attributes**
  - Declaring attributes in attrs.xml
  - Setting a custom namespace in layout file
  - Using the custom attributes in layout file
  - Extracting values from AttributeSet in View constructor

Custom Components

Custom Components Overview

- **Idea**
  - An extension of View that has drawing and/or event handlers
    - Examples: stop light that acts like checkbox, speedometer dial that acts like slider
- **What's new**
  - In sections on drawing, we already made custom View classes with onDraw and sometimes touch event handlers. So what else do we need?
    - Smart size calculations with onMeasure
      - In fixed-size regions, need to resize content to fit
      - In variable-size regions, need to calculate needed size
    - Custom XML attributes in XML file
      - Pass app-specific attributes from the layout file

Using onMeasure

onMeasure

- **Idea**
  - In fixed-size regions, resize content to fit
  - In variable-size regions, calculate needed size
- **Java syntax**

```
protected void onMeasure(int widthMeasureSpec,
                         int heightMeasureSpec) {
  // Assign width and height variables as follows:
  //   - If fixed-size (getMode is EXACTLY), use getSize
  //   - Otherwise set to desired size
  setMeasuredDimension(width, height);
}
```


Determining Region Type

- **Fixed-size region**
  - MeasureSpec.getMode(measureSpec) == MeasureSpec.EXACTLY
    - Set size to MeasureSpec.getValue(measureSpec)
- **Variable-size region**
  - MeasureSpec.getMode(measureSpec) != MeasureSpec.EXACTLY
    - Calculate space desired. Don't forget padding.
- **Variable-size region with max**
  - MeasureSpec.getMode(measureSpec) == MeasureSpec.AT_MOST
    - Return smaller of above two values

Java Template: onMeasure

```
protected void onMeasure(int widthMeasureSpec,
                         int heightMeasureSpec) {
    int width = measureDim(widthMeasureSpec);
    int height = measureDim(heightMeasureSpec);
    adjustThingsToFit(width, height);
    setMeasuredDimension(width, height);
}
```


Java Template: Helper Method Called by onMeasure

```
private int measureDim(int measureSpec) {
    int result = 0;
    int specMode = MeasureSpec.getMode(measureSpec);
    int specSize = MeasureSpec.getSize(measureSpec);
    if (specMode == MeasureSpec.EXACTLY) {
       result = specSize;
    } else {
       result = someCalculationOfSpaceNeeded();
       if (specMode == MeasureSpec.AT_MOST) {
          result = Math.min(result, specSize);
       }
    }
    return(result);
  }
```

Example: View that Draws Text in Circle

Example Summary

- **Idea**
  - Make View that draws text rotated around a central point
    - But this time, don't just guess on font size as before
- **For fixed-size regions**
  - Adjust font size to fit smaller of width and height
- **For variable-sized regions**
  - Take twice the string length in the default text size. Add in the View padding. Make that the desired size for the smaller dimension.

View: General Class Structure

```
public class CircleTextView extends View {
    private int mDefaultTextSize = 20;
    private int mTextColor = Color.BLUE;
    private Paint mPaint = makePaint(mTextColor);
    private Paint mTestPaint = makePaint(mTextColor);
    private String mMessage = "Android";


    public CircleTextView(Context context) {
        super(context);
    }

    public CircleTextView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    ...
}
```


View: onDraw

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    int viewWidth = getWidth();
    int viewHeight = getHeight();
    canvas.translate(viewWidth/2, viewHeight/2);
    for(int i=0; i<10; i++) {
        canvas.drawText(mMessage, 0, 0, mPaint);
        canvas.rotate(36);
    }
}
```

This is the same basic onDraw as in the coordinate transformations section of the second tutorial on drawing in Android.

View: onMeasure and Simple Helper Functions

```
@Override
protected void onMeasure(int widthMeasureSpec,
                         int heightMeasureSpec) {
    int width = measureWidth(widthMeasureSpec);
    int height = measureHeight(heightMeasureSpec);
    int textWidth =
            width - getPaddingLeft() - getPaddingRight();
    int textHeight =
            height - getPaddingTop() - getPaddingBottom();
    adjustTextSizeToFit(Math.min(textWidth, textHeight));
    setMeasuredDimension(width, height);
}
private int measureWidth(int measureSpec) {
    return(measureText(measureSpec, getPaddingLeft(),
                       getPaddingRight()));
}
private int measureHeight(int measureSpec) {
    return(measureText(measureSpec, getPaddingTop(),
                       getPaddingBottom()));
}
```


View: Main Measuring Helper Function

```
private int measureText(int measureSpec,
                        int padding1, int padding2) {
    int result = 0;
    int specMode = MeasureSpec.getMode(measureSpec);
    int specSize = MeasureSpec.getSize(measureSpec);
    if (specMode == MeasureSpec.EXACTLY) {
        // We were told how big to be,
        // so set text size to make it fit
        result = specSize;
    } else {
        // Measure the text: twice text size plus padding
        result = 2*(int)mPaint.measureText(mMessage) +
                 padding1 + padding2;
        if (specMode == MeasureSpec.AT_MOST) {
            // Respect AT_MOST value if that was
            // what is called for by measureSpec
            result = Math.min(result, specSize);
        }
    }
    return(result);
}
```

View: Helper Function to Adjust Size

```
private void adjustTextSizeToFit(int availableLength) {
    int textSize = 0;
    for(int i=1; i<=10; i++) {
        textSize = i * 10;
        mTestPaint.setTextSize(textSize);
        int requiredLength =
            2 * (int)mTestPaint.measureText(mMessage);
        if (requiredLength > availableLength) {
            break;
        }
    }
    mPaint.setTextSize(Math.max(10, textSize - 10));
}
```


View: Auxiliary Helper Function

```
private Paint makePaint(int color) {
    // Angled lines look much better with antialiasing
    Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
    p.setColor(color);
    p.setTextSize(mDefaultTextSize); // Default size
                                     // (e.g., for
                                     // wrap_content)
    return(p);
}
```

Layout File for Example (activity_measurements.xml)

```
<LinearLayout ...
    android:orientation="vertical" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <view
            class="com.coreservlets.customcomponents.CircleTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@android:color/holo_orange_light" />
        <view
            class="com.coreservlets.customcomponents.CircleTextView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@android:color/holo_blue_light"  />
    </LinearLayout>
    <view
        class="com.coreservlets.customcomponents.CircleTextView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```


Activity for Example

```
public class MeasurementsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_measurements);
    }
}
```

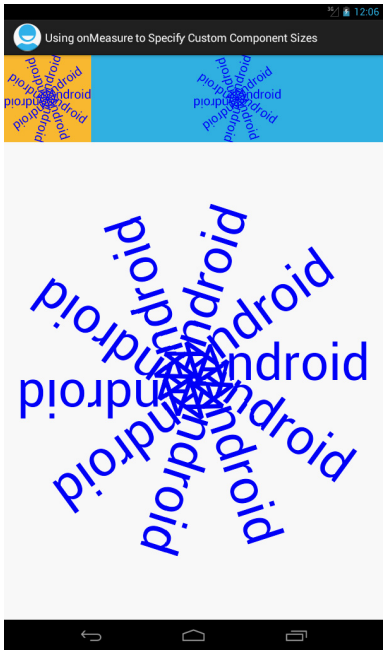
Overall Main Layout File (main.xml)

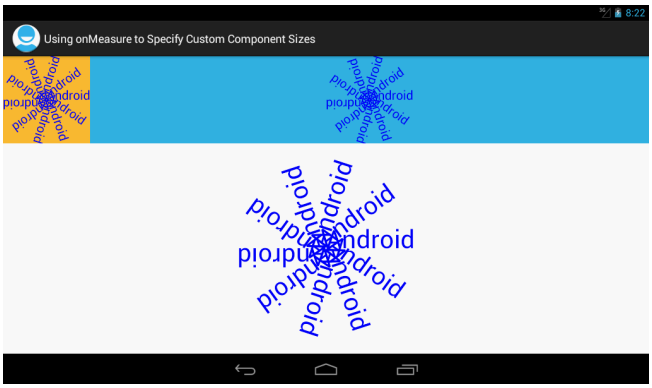
```
<LinearLayout ... android:orientation="vertical">
    <Button
        android:onClick="launchMeasurementsActivity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_label_activity_measurements" />
    <Button
        android:onClick="launchCustomAttributesActivity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_label_activity_custom_attributes" />
</LinearLayout>
```


Overall Main Activity

```
public class CustomComponentsLauncher extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void launchMeasurementsActivity(View clickedButton) {
        Intent activityIntent =
                new Intent(this, MeasurementsActivity.class);
        startActivity(activityIntent);
    }

    public void launchCustomAttributesActivity(View clickedButton) {
        Intent activityIntent =
                new Intent(this, CustomAttributesActivity.class);
        startActivity(activityIntent);
    }

}
```

Results





Custom Attributes

Custom Attributes in XML File

- **Idea**
  - Custom Views may need to have special-purpose values specified in the layout file
- **Steps**
  - Declare the attributes in res/values/attrs.xml
    - For declare-styleable, give class name of custom View
  - In layout file, declare custom namespace
    - xmlns:yourprefix= "http://schemas.android.com/apk/res/your.package"
  - In layout file, use custom attributes
    - yourprefix:your_attribute="your value"
  - In View constructor, extract values from AttributeSet
    - Obtain TypedArray
    - Use getInt, getFloat, getString, getColor, getDrawable, etc.

Step 1: Declare Attributes in res/values/attrs.xml

- **Basic structure**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <declare-styleable name="YourCustomView">
      <attr name="attribute_1" format="integer"/>
      <attr name="attribute_2" format="float"/>
      <attr name="attribute_3" format="string"/>
   </declare-styleable>
</resources>
```

- **Legal values for "format" attribute**
  - boolean, color, dimension, enum, flag, float, fraction, integer, string, reference

Calling this file attrs.xml is customary, but not technically required.

Step 2: Declare Custom Namespace in Layout File

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:yourprefix="http://schemas.android.com/apk/res/your.package"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
  …
  <view class="your.package.YourCustomView" … />
  …
</LinearLayout>
```


Step 3: Use Custom Attributes in Layout File

- **Basic format**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:yourprefix="http://schemas.android.com/apk/res/your.package"
  … >
  …
  <view
    class="your.package.YourCustomView"
    android:layout_width="…"
    android:layout_height="…"
    yourprefix:attribute_1="123"
    yourprefix:attribute_2="4.56"
    yourprefix:attribute_3="some string" />
  …
</LinearLayout>
```

Matches actual class name. Also matches "name" attribute of declare-styleable in attrs.xml.

Attribute names match the "name" attribute of the attr elements in attrs.xml.

Step 4: Extract Attribute Values in View Constructor

- **Basic format**

```
public YourCustomView(Context context, AttributeSet attrs) {
    super(context, attrs);
    TypedArray attributeArray =
        getContext().obtainStyledAttributes(attrs, R.styleable.YourCustomView);
    int att1Value =
        attributeArray.getInt(R.styleable.YourCustomView_attribute_1,
                              someDefaultInt);
    float att2Value =
        attributeArray.getFloat(R.styleable.YourCustomView_attribute_2,
                                someDefaultFloat);
    String att3Value =
        attributeArray.getString(R.styleable.YourCustomView_attribute_3);
    if (att3Value == null) { att3Value = someDefaultString; }
}
```

Instead of doing this initialization *directly* in the constructor, it is common to pass the AttributeSet to a separate initialization method that does this work.

Example: View with Custom Attributes

Example Summary

- **Idea**
  - Make View that draws text rotated around a central point
    - But this time, let layout file specify extra attributes to customize the look
- **Custom attributes**
  - default_text_size
    - The font size to use for variable-size regions. (For fixed-size regions, this is ignored and font is stretched to fit available space).
  - text_color
    - The color with which to draw the message.
  - text_messsage
    - The actual message that is drawn in a circle

View: General Class Structure and Constructors

```
public class CustomizableCircleTextView extends View {
    private int mDefaultTextSize = 20;
    private int mTextColor = Color.BLUE;
    private Paint mPaint;
    private Paint mTestPaint = makePaint(mTextColor);
    private static final String DEFAULT_MESSAGE = "Android";
    private String mMessage = DEFAULT_MESSAGE;

    public CustomizableCircleTextView(Context context) {
        super(context);
        mPaint = makePaint(mTextColor);
    }

    public CustomizableCircleTextView(Context context,
                                      AttributeSet attrs) {
        super(context, attrs);
        initializeCustomAttributes(attrs);
        mPaint = makePaint(mTextColor);
    }
```

View: Extracting Values of Custom Attributes

```
private void initializeCustomAttributes(AttributeSet attrs) {
    TypedArray attributeArray =
       getContext().obtainStyledAttributes(attrs,
                                           R.styleable.CustomizableCircleTextView);
    mDefaultTextSize =
       attributeArray.getInt(R.styleable.CustomizableCircleTextView_default_text_size,
                             mDefaultTextSize);
    mTextColor =
       attributeArray.getInt(R.styleable.CustomizableCircleTextView_text_color,
                             mTextColor);
    mMessage =
       attributeArray.getString(R.styleable.CustomizableCircleTextView_text_message);
    if (mMessage == null) {
       mMessage = DEFAULT_MESSAGE;
    }
}
```


View: Adjusting the Font Size

```
private void adjustTextSizeToFit(int availableLength) {
    int textSize = 0;
    for(int i=1; i<=10; i++) {
        textSize = i * 10;
        mTestPaint.setTextSize(textSize);
        int requiredLength =
            2 * (int)mTestPaint.measureText(mMessage);
        if (requiredLength > availableLength) {
            break;
        }
    }
    mPaint.setTextSize(Math.max(10, textSize - 10));
}
```

Unmodified Code

- **Point**
  - Many methods don't have to change from the previous example, except for perhaps minor changes in variable names.
    - The difference is that the text of mMessage and the color and font size in mPaint may have changed based on attributes in the layout file. But this all happens before many of the methods are called.
- **Unchanged methods from previous example**
  - onDraw
  - onMeasure
  - Helper methods for onDraw
    - measureHeight
    - measureWidth
    - measureText
  - makePaint

Custom Attributes File (res/values/attrs.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="CustomizableCircleTextView">
        <attr name="default_text_size" format="integer"/>
        <attr name="text_color" format="integer"/>
        <attr name="text_message" format="string"/>
    </declare-styleable>
</resources>
```

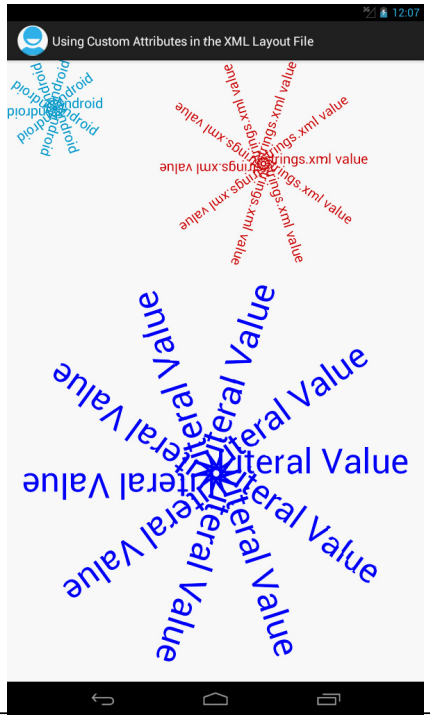
Layout File for Example (activity_custom_attributes.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:coreservlets="http://schemas.android.com/apk/res/com.coreservlets.customcomponents"
  …
  android:orientation="vertical" >
  <LinearLayout  android:layout_width="match_parent" android:layout_height="wrap_content" >
    <view
      class="com.coreservlets.customcomponents.CustomizableCircleTextView"
      …
      coreservlets:text_color="@android:color/holo_blue_dark" />
    <view
      class="com.coreservlets.customcomponents.CustomizableCircleTextView"
      …
      coreservlets:default_text_size="40"
      coreservlets:text_color="@android:color/holo_red_dark"
      coreservlets:text_message="@string/message_activity_custom_attributes" />
  </LinearLayout>
  <view
    class="com.coreservlets.customcomponents.CustomizableCircleTextView"
    …
    coreservlets:text_message="Literal Value"  />
</LinearLayout>
```


Activity for Example

```
public class CustomAttributesActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_custom_attributes);
    }
}
```

Results



Wrap-Up

References

- **onMeasure**
  - http://developer.android.com/guide/topics/ui/custom-components.html
- **Custom attributes**
  - http://developer.android.com/training/custom-views/create-view.html
  - http://kevindion.com/2011/01/custom-xml-attributes-for-android-widgets/
  - http://blog.infidian.com/2008/05/02/android-tutorial-42-passing-custom-variables-via-xml-resource-files/
  - http://ficklemind.blogspot.com/2011/03/how-to-compose-attrsxml-in-android.html

Summary

- **onMeasure**
  - In fixed-size regions (MeasureSpec.EXACTLY), resize
  - In variable-size regions, calculate needed size
  - return(setMeasuredDimension(calcWidth, calcHeight));
- **Custom attributes**
  - Declare the attributes in res/values/attrs.xml
    - For declare-styleable, give class name of custom View
  - In layout file, declare custom namespace
    - xmlns:yourprefix="http://.../apk/res/your.package"
  - In layout file, use custom attributes
    - yourprefix:your_attribute="your value"
  - In View constructor, extract values from AttributeSet
    - Obtain TypedArray, then use getInt/getString, etc.

Questions?