Android - Creating App Widgets

Widgets are little applications which can be placed on the *home screen* or the *lock screen*
The widget has the permissions of the application.
Widget use `RemoteViews` to create their user interface. A `RemoteView` can be executed by another process with the same permissions as the original application (pending intent is needed).
The user interface for a *Widget* is defined by a broadcast receiver. This receiver inflates its layout into an object of type `RemoteViews`. This object is delivered to Android, which hands it over the home screen application.

Steps to create a Widget

To create a widget, you:

- Define the layout xml file
- Create an XML file (`AppWidgetProviderInfo in the res/xml folder`) which describes the properties of the widget, e.g. size or the fixed update frequency.
- Create a `BroadcastReceiver` which is used to build the user interface of the widget.
- Enter the *Widget* configuration in the *AndroidManifest.xml* file.

creating the widget layout

A widget is restricted in the View classes it can use. As layouts you can use the FrameLayout, LinearLayout and RelativeLayout classes. As views you can use AnalogClock, Button, Chromometer, ImageButton, ImageView, ProgressBar and TextView.
As of Android 3.0 more views are available: GridView, ListView, StackView, ViewFlipperand AdapterViewFlipper.
<u>The only interaction that is possible with the views of a widget is via an OnClickListener event</u>. This OnClickListener can be registered on a widget and is triggered by the user.
The simplest layout you can create will be a LinearLayout containing an ImageButton. Don't forget to give that button an id and a source image that will be your widget's appearance. You can also create

more complex layouts containing a few widgets(for example an image and text for weather updates and more). The widget layout is the same as any other layout.

Creating configuration file

The configuration file is located in the res/xml folder and contains the widget size, widget update interval(or 0 if you don't the views to be updated), the widget initial layout file and more.
Before Android 3.1 a widget always took a fixed amount of cells on the home screen. A cell is usually used to display the icon of one application. As a calculation rule you should define the size of the widget with the formula: ((Number of columns / rows) * 74) - 2. These are device independent pixels and the -2 is used to avoid rounding errors.
As of Android 3.1 a widget can be flexible in size, e.g., the user can make it larger or smaller. To enable this for widget, you can use the android:resizeMode="horizontal|vertical".
additional attributes are previewImage that specifies a preview of what the app widget will look like after it's configured, which the user sees when selecting the app widget. If not supplied, the user instead sees your application's launcher icon. and widgetCategory attribute declares whether your App Widget can be displayed on the home screen (home_screen), the lock screen (keyguard), or both(use the bitwise-ored). Only Android versions lower than 5.0 support lock-screen widgets. For Android 5.0 and higher, only home_screen is valid.

configuration file example:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen">
</appwidget-provider>
```

Creating the Broadcast receiver for the widget

Your BroadcastReceiver implementation typically extends the AppWidgetProvider class.
The AppWidgetProvider class implements the onReceive() method, extracts the required information and calls the following widget life cycle methods, So there is no need for to override the Broadcast receiver onReceive method but just use the following methods.

Table 1. Life cycle method

Method	Description
onEnabled()	Called the first time an instance of your widget is added to the home screen.
onDisabled()	Called once the last instance of your widget is removed from the home screen.
onUpdate()	Called for every update of the widget. Contains the ids of appWidgetIds for which an update is needed.
onDeleted()	Widget instance is removed from the home screen.

The most important AppWidgetProvider callback is **onUpdate**() because it is called when each App Widget is added to a host. If your App Widget accepts any user interaction events, then you need to register the event handlers in this callback. For example if your appwidget contains an imagebutton here we can register the click listener to it. If your App Widget doesn't create temporary files or databases, or perform other work that requires clean-up, then onUpdate() may be the only callback method you need to define. For example, if you want an App Widget with a button that launches an Activity when clicked, you could use the following implementation of AppWidgetProvider:

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {

    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
    final int N = appWidgetIds.length;

    // Perform this loop procedure for each App Widget that belongs to this provider
    for (int i=0; i<N; i++) {
        int appWidgetId = appWidgetIds[i];

        // Create an Intent to launch ExampleActivity
        Intent intent = new Intent(context, ExampleActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);

        // Get the layout for the App Widget and attach an on-click listener
        // to the button
        RemoteViews views = new RemoteViews(context.getPackageName(),
        R.layout.appwidget_provider_layout);
        views.setOnClickPendingIntent(R.id.button, pendingIntent);

        // Tell the AppWidgetManager to perform an update on the current app widget
        appWidgetManager.updateAppWidget(appWidgetId, views);
    }
}
```

Declaring an App Widget in the Manifest

The last thing we need to do is to declare the widget's broadcastreceiver (AppWidgetProvider) in the application's AndroidManifest.xml file. For example:

```
<receiver android:name="ExampleAppWidgetProvider" >
  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <meta-data android:name="android.appwidget.provider"
```

```
            android:resource="@xml/example_appwidget_info" />
</receiver>
```

The two things we need to set is:

1. receiver name is the name of AppwidgetProvider implementation (The broadcast receiver).
2. The xml configuration file in the android:resource attribute

References

1. Android (Home screen) Widgets - Tutorial by Lars Vogel
2. Android developer guide - App Widgets