Mobile Programming

[week 06]

[App Widgets]

Hilmy A. T.

hilmi.tawakal@gmail.com

Introduction

- App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates.
- These views are referred to as Widgets in the user interface, and you can publish one with an App Widget provider.
- An application component that is able to hold other App Widgets is called an App Widget host.

Introduction

- Home screen widgets are views that can be displayed on a home page and updated frequently.
- As a view, a widget's look and feel is defined through a layout XML file.
- For a widget, in addition to the layout of the view, you will need to define how much space the view of the widget will need on the home screen.

Creating Widgets

To create an App Widget, you need the following:

- AppWidgetProviderInfo object:

  Describes the metadata for an App Widget, such as the App Widget's layout, update frequency, and the AppWidgetProvider class. This should be defined in XML.

- AppWidgetProvider class implementation:

  Defines the basic methods that allow you to programmatically interface with the App Widget, based on broadcast events. Through it, you will receive broadcasts when the App Widget is updated, enabled, disabled and deleted.

- View layout:

  Defines the initial layout for the App Widget, defined in XML.

Creating Widgets: Configure at AndroidManifest.xml

First, declare the AppWidgetProvider class in your application's AndroidManifest.xml file. For example:

```
<receiver android:name=".MyWidgetProvider">
    <meta-data android:name="android.appwidget.provider"
               android:resource="@xml/my_appwidget_provider_info"/>
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE"/>
    </intent-filter>
</receiver>
```

Creating Widgets: Configure at AndroidManifest.xml

- The <receiver> element requires the android:name attribute, which specifies the AppWidgetProvider used by the App Widget.
- The <intent-filter> element must include an <action> element with the android:name attribute. This attribute specifies that the AppWidgetProvider accepts the ACTION_APPWIDGET_UPDATE broadcast. This is the only broadcast that you must explicitly declare. The AppWidgetManager automatically sends all other App Widget broadcasts to the AppWidgetProvider as necessary.

Creating Widgets: Configure at AndroidManifest.xml

- The <meta-data> element specifies the AppWidgetProviderInfo resource and requires the following attributes:
  - android:name

    Specifies the metadata name. Use android.appwidget.provider to identify the data as the AppWidgetProviderInfo descriptor.
  - android:resource

    Specifies the AppWidgetProviderInfo resource location.

Creating Widgets: AppWidgetProviderInfo

- The AppWidgetProviderInfo defines the essential qualities of an App Widget, such as its minimum layout dimensions, its initial layout resource, how often to update the App Widget, and (optionally) a configuration Activity to launch at create-time.
- Define the AppWidgetProviderInfo object in an XML resource using a single <appwidget-provider> element and save it in the project's res/xml/ folder. For example:

Creating Widgets: AppWidgetProviderInfo

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

Creating Widgets: AppWidgetProviderInfo

Here's a summary of the <appwidget-provider> attributes:

- The values for the minWidth and minHeight attributes specify the minimum amount of space the App Widget consumes by default.
- The minResizeWidth and minResizeHeight attributes specify the App Widget's absolute minimum size.

Creating Widgets: AppWidgetProviderInfo

- The updatePeriodMillis attribute defines how often the App Widget framework should request an update from the AppWidgetProvider by calling the onUpdate() callback method.
- The initialLayout attribute points to the layout resource that defines the App Widget layout.
- The configure attribute defines the Activity to launch when the user adds the App Widget, in order for him or her to configure App Widget properties.

Creating Widgets: AppWidgetProviderInfo

- The previewImage attribute specifies a preview of what the app widget will look like after it's configured, which the user sees when selecting the app widget.
- The autoAdvanceViewId attribute specifies the view ID of the app widget subview that should be auto-advanced by the widget's host.
- The resizeMode attribute specifies the rules by which a widget can be resized.

Creating Widgets: AppWidgetProviderInfo

- The minResizeHeight attribute specifies the minimum height (in dps) to which the widget can be resized.
- The minResizeWidth attribute specifies the minimum width (in dps) to which the widget can be resized.
- The widgetCategory attribute declares whether your App Widget can be displayed on the home screen, the lock screen (keyguard), or both.

Creating Widgets: AppWidgetProviderInfo

- The initialKeyguardLayout attribute points to the layout resource that defines the lock screen App Widget layout.

See the AppWidgetProviderInfo class for more information on the attributes accepted by the <appwidget-provider> element.

Creating Widgets: App Widget Layout

- You must define an initial layout for your App Widget in XML and save it in the project's res/layout/ directory.
- Creating the App Widget layout is simple if you're familiar with Layouts. However, you must be aware that App Widget layouts are based on RemoteViews, which do not support every kind of layout or view widget.

Creating Widgets: AppWidgetProvider Class

- The AppWidgetProvider class extends BroadcastReceiver as a convenience class to handle the App Widget broadcasts.
- The AppWidgetProvider receives only the event broadcasts that are relevant to the App Widget, such as when the App Widget is updated, deleted, enabled, and disabled.

Creating Widgets: AppWidgetProvider Class

- When these broadcast events occur, the AppWidgetProvider receives the following method calls:
  - OnUpdate()
  - OnAppWidgetOptionsChanged()
  - onDeleted(Context, int[])
  - onEnabled(Context)
  - onDisabled(Context)
  - onReceive(Context, Intent)

Creating Widgets: AppWidgetProvider Class

onUpdate()

- This is called to update the App Widget at intervals defined by the updatePeriodMillis attribute in the AppWidgetProviderInfo.
- This method is also called when the user adds the App Widget, so it should perform the essential setup, such as define event handlers for Views and start a temporary Service, if necessary.
- However, if you have declared a configuration Activity, this method is not called when the user adds the App Widget, but is called for the subsequent updates. It is the responsibility of the configuration Activity to perform the first update when configuration is done

Creating Widgets: AppWidgetProvider Class

onAppWidgetOptionsChanged()

- This is called when the widget is first placed and any time the widget is resized.
- You can use this callback to show or hide content based on the widget's size ranges. You get the size ranges by calling getAppWidgetOptions(), which returns a Bundle that includes the following:
  - OPTION_APPWIDGET_MIN_WIDTH
  - OPTION_APPWIDGET_MIN_HEIGHT
  - OPTION_APPWIDGET_MAX_WIDTH
  - OPTION_APPWIDGET_MAX_HEIGHT

Creating Widgets: AppWidgetProvider Class

onDeleted(Context, int[])

- This is called every time an App Widget is deleted from the App Widget host.

onEnabled(Context)

- This is called when an instance the App Widget is created for the first time. For example, if the user adds two instances of your App Widget, this is only called the first time. If you need to open a new database or perform other setup that only needs to occur once for all App Widget instances, then this is a good place to do it.

Creating Widgets: AppWidgetProvider Class

onDisabled(Context)

- This is called when the last instance of your App Widget is deleted from the App Widget host. This is where you should clean up any work done in onEnabled(Context), such as delete a temporary database.

onReceive(Context, Intent)

- This is called for every broadcast and before each of the above callback methods. You normally don't need to implement this method because the default AppWidgetProvider implementation filters all App Widget broadcasts and calls the above methods as appropriate.

App Widget Configuration Activity

- If you would like the user to configure settings when he or she adds a new App Widget, you can create an App Widget configuration Activity.
- This Activity will be automatically launched by the App Widget host and allows the user to configure available settings for the App Widget at create-time, such as the App Widget color, size, update period or other functionality settings.

App Widget Configuration Activity

- The configuration Activity should be declared as a normal Activity in the Android manifest file.
- However, it will be launched by the App Widget host with the ACTION_APPWIDGET_CONFIGURE action, so the Activity needs to accept this Intent.

```
<activity android:name=".ExampleAppWidgetConfigure">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE"/>
    </intent-filter>
</activity>
```

App Widget Configuration Activity

- Also, the Activity must be declared in the AppWidgetProviderInfo XML file, with the android:configure attribute.

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    ... >
</appwidget-provider>
```

App Widget Configuration Activity

- Notice that the Activity is declared with a fully-qualified namespace, because it will be referenced from outside your package scope.
- That's all you need to get started with a configuration Activity. Now all you need is the actual Activity.

App Widget Configuration Activity

There are, however, two important things to remember when you implement the Activity:

- The App Widget host calls the configuration Activity and the configuration Activity should always return a result. The result should include the App Widget ID passed by the Intent that launched the Activity (saved in the Intent extras as EXTRA_APPWIDGET_ID).
- The onUpdate() method will not be called when the App Widget is created (the system will not send the ACTION_APPWIDGET_UPDATE broadcast when a configuration Activity is launched). It is the responsibility of the configuration Activity to request an update from the AppWidgetManager when the App Widget is first created. However, onUpdate() will be called for subsequent updates—it is only skipped the first time.

Updating the App Widget from the configuration Activity

- When an App Widget uses a configuration Activity, it is the responsibility of the Activity to update the App Widget when configuration is complete. You can do so by requesting an update directly from the AppWidgetManager.
- Here's a summary of the procedure to properly update the App Widget and close the configuration Activity:

Updating the App Widget from the configuration Activity

1. First, get the App Widget ID from the Intent that launched the Activity:

```
Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
    mAppWidgetId = extras.getInt(
            AppWidgetManager.EXTRA_APPWIDGET_ID,
            AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

Updating the App Widget from the configuration Activity

2.Perform your App Widget configuration.

3.When the configuration is complete, get an instance of the AppWidgetManager by calling getInstance(Context):

```
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

Updating the App Widget from the configuration Activity

4. Update the App Widget with a RemoteViews layout by calling updateAppWidget(int, RemoteViews):

```
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.example_appwidget);
appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

Updating the App Widget from the configuration Activity

5.Finally, create the return Intent, set it with the Activity result, and finish the Activity:

```
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);
finish();
```

Setting a Preview Image

- Android 3.0 introduces the previewImage field, which specifies a preview of what the app widget looks like.
- This preview is shown to the user from the widget picker.
- If this field is not supplied, the app widget's icon is used for the preview.

Setting a Preview Image

This is how you specify this setting in XML:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
  ...
  android:previewImage="@drawable/preview">
</appwidget-provider>
```

Setting a Preview Image

- To help create a preview image for your app widget (to specify in the previewImage field), the Android emulator includes an application called "Widget Preview."
- To create a preview image, launch this application, select the app widget for your application and set it up how you'd like your preview image to appear, then save it and place it in your application's drawable resources.

Enabling App Widgets on the Lockscreen

- Android 4.2 introduces the ability for users to add widgets to the lock screen.
- To indicate that your app widget is available for use on the lock screen, declare the android:widgetCategory attribute in the XML file that specifies your AppWidgetProviderInfo.
- This attribute supports two values: "home_screen" and "keyguard". An app widget can declare support for one or both.

Enabling App Widgets on the Lockscreen

- By default, every app widget supports placement on the Home screen, so "home_screen" is the default value for the android:widgetCategory attribute.
- If you want your app widget to be available for the lock screen, add the "keyguard" value:

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:widgetCategory="keyguard|home_screen">
</appwidget-provider>
```

Question?