Grid View

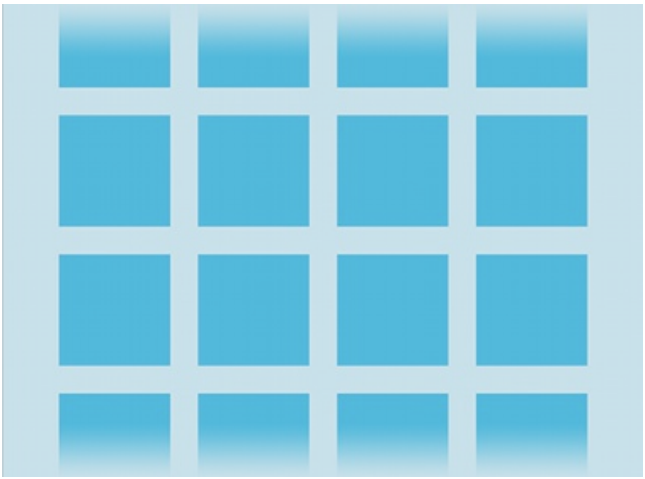
In this document

Key classes

- `GridView`
- `ImageView`
- `BaseAdapter`
- `AdapterView.OnItemClickListener`

`GridView` is a `ViewGroup` that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a `ListAdapter`.

For an introduction to how you can dynamically insert views using an adapter, read Building Layouts with an Adapter.



Example

In this tutorial, you'll create a grid of image thumbnails. When an item is selected, a toast message will display the position of the image.

1. Start a new project named *HelloGridView*.
2. Find some photos you'd like to use, or download these sample images. Save the image files into the project's `res/drawable/` directory.
3. Open the `res/layout/main.xml` file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

This `GridView` will fill the entire screen. The attributes are rather self explanatory. For more information about valid attributes, see the `GridView` reference.

4. Open `HelloGridView.java` and insert the following code for the `onCreate()` method:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v,
                int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position,
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

After the `main.xml` layout is set for the content view, the `GridView` is captured from the layout with `findViewById(int)`. The `setAdapter()` method then sets a custom adapter (`ImageAdapter`) as the source for all items to be displayed in the grid. The `ImageAdapter` is created in the next step.

To do something when an item in the grid is clicked, the `setOnItemClickListener()` method is passed a new `AdapterView.OnItemClickListener`. This anonymous instance defines the `onItemClick()` callback method to show a `Toast` that displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

5. Create a new class called `ImageAdapter` that extends `BaseAdapter`:

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {
            // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = {
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7
    };
}
```

First, this implements some required methods inherited from `BaseAdapter`. The constructor and `getCount()` are self-explanatory. Normally, `getItem(int)` should return the actual object at the specified position in the adapter, but it's ignored for this example. Likewise, `getItemId(int)` should return the row id of the item, but it's not needed here.

The first method necessary is `getView()`. This method creates a new `View` for each image added to the `ImageAdapter`. When this is called, a `View` is passed in, which is normally a recycled object (at least after this has been called once), so there's a check to see if the object is null. If it *is* null, an `ImageView` is instantiated and configured with desired properties for the image presentation:

- `setLayoutParams(ViewGroup.LayoutParams)` sets the height and width for the View—this ensures that, no matter the size of the drawable, each image is resized and cropped to fit in these dimensions, as appropriate.
- `setScaleType(ImageView.ScaleType)` declares that images should be cropped toward the center (if necessary).
- `setPadding(int, int, int, int)` defines the padding for all sides. (Note that, if the images have different aspect-ratios, then less padding will cause more cropping of the image if it does not match the dimensions given to the ImageView.)

If the `View` passed to `getView()` is *not* null, then the local `ImageView` is initialized with the recycled `View` object.

At the end of the `getView()` method, the `position` integer passed into the method is used to select an image from the `mThumbIds` array, which is set as the image resource for the `ImageView`.

All that's left is to define the `mThumbIds` array of drawable resources.

6. Run the application.

Try experimenting with the behaviors of the `GridView` and `ImageView` elements by adjusting their properties. For example, instead of using `setLayoutParams(ViewGroup.LayoutParams)`, try using `setAdjustViewBounds(boolean)`.