

Mobile Application Development

User Interface

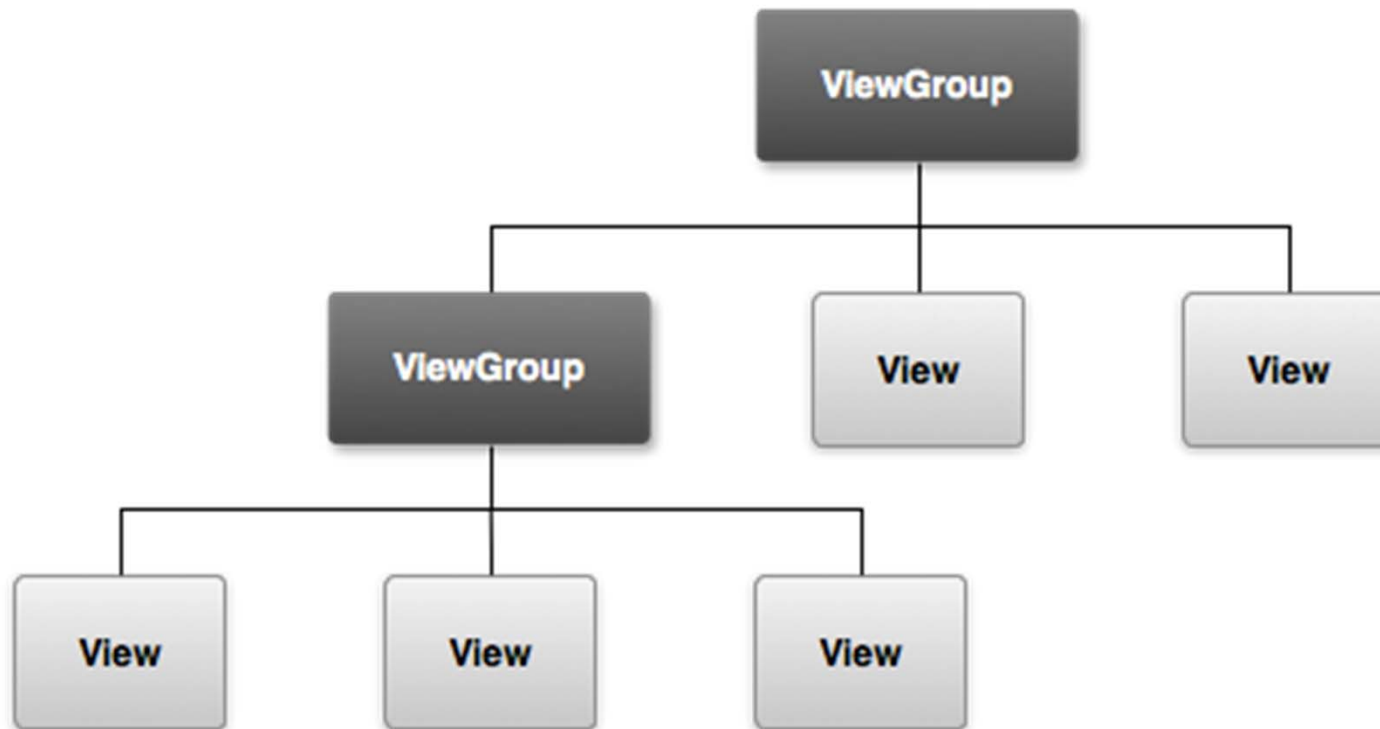
Layouts

User Interface

- All user interface elements in an Android app are built using **View** and **ViewGroup** objects.
 - A **View** is an object that draws something on the screen that the user can interact with (such as buttons and text fields).
 - A **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface (such as a linear or relative layout).

User Interface

- The user interface for each component of your app is defined using a [hierarchy of View and ViewGroup objects](#).



User Interface

- In addition to using View and ViewGroup objects, you can also use several **app components** that offer a standard UI layout for which you simply need to define the content.
- For Example:
 - Action Bar,
 - Dialogs, and
 - Status Notifications.

LAYOUTS

Layouts

- A layout defines the visual structure for a user interface, such as the UI for an activity.
- You can declare a layout in two ways:
 - **Declare UI elements in XML.** Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses.
 - **Instantiate layout elements at runtime.** Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Layouts

- The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI.
- The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior.
 - You can modify or adapt it without having to modify your source code
 - You can create XML layouts for different screen orientations, different device screen sizes, and different languages.

XML Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```


XML Layout

- After you've declared your layout in XML, save the file with the `.xml` extension, in your Android project's `res/layout/` directory, so it will properly compile.

Load the XML Resource

- When you compile your application, each XML layout file is compiled into a View resource.
- You should load the layout resource from your application code, in your `Activity.onCreate()` callback implementation.
- You can do this by calling `setContentView()` – and passing it the reference to your layout resource in the form of:
`R.layout.layout_file_name`
- The `onCreate()` callback method in your Activity is called by the Android framework when your Activity is launched

Load the XML Resource

- For example, if your XML layout is saved as `main_layout.xml`, you would load it for your Activity like this:

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

Remember Accessing Resources:

`R.<resource_type>.<resource_name>`

Attributes

- Every **View** and **ViewGroup** object supports their own variety of XML attributes.
- Some attributes are **specific to a View** object, but some are **common to all View objects**, because they are inherited from the **root View** class (like the “id” attribute).
- And, other attributes are considered “**layout parameters**”, which are attributes that describe certain layout orientations of the View object, as **defined by that object's parent ViewGroup** object.

The “id” Attribute

- Any View object may have an **integer ID** associated with it, **to uniquely identify the View** within the tree.
- This is an XML attribute common to all View objects (defined by the View class) and you will use it very often.
- The syntax for an ID, inside an XML tag is: **android:id="@+id/my_button"**
- The **at-symbol** (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The **plus-symbol** (+) means that this is a new resource name that must be created and added to our resources (in the **R.java** file).

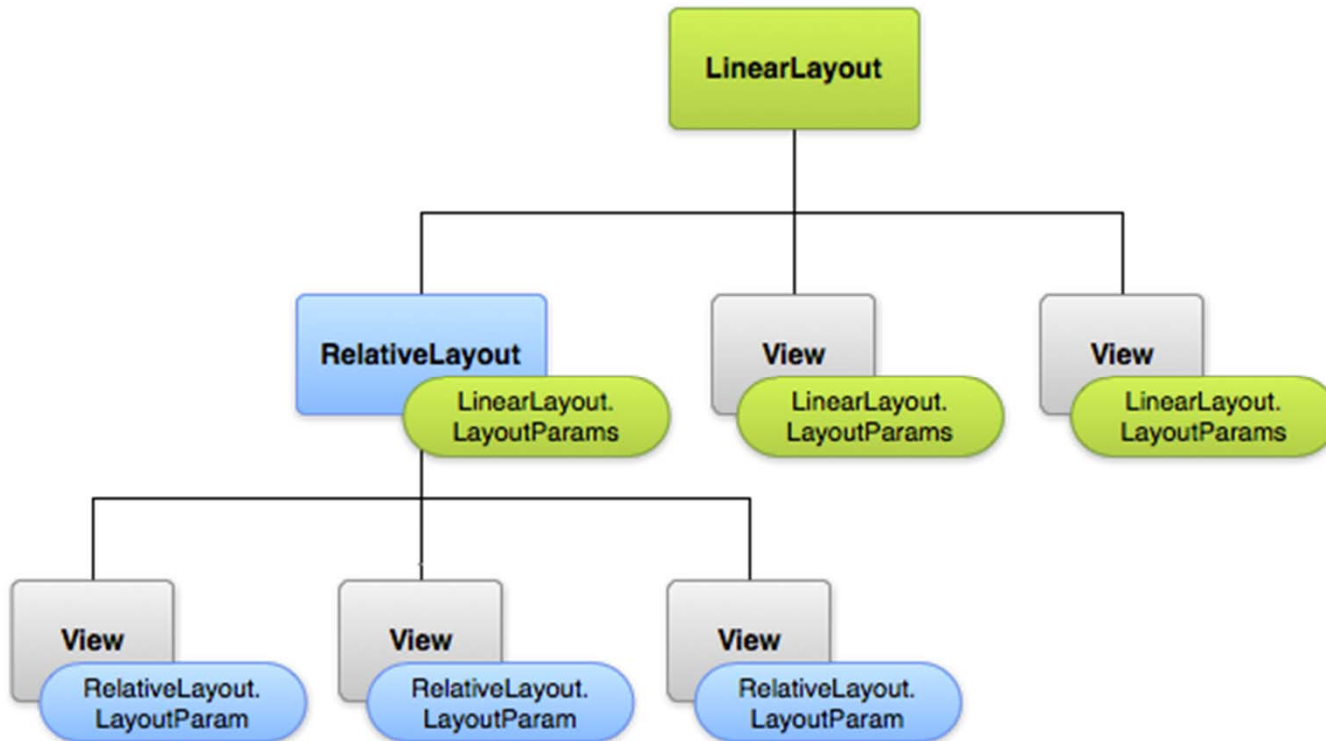
The “id” Attribute

- When you need to reference an existing resource ID, you do not need the plus-symbol:

`android:id="@id/my_button"`

Layout Parameters

- XML layout attributes named `layout_xyz` define layout parameters for the View that are appropriate for the ViewGroup in which it resides.



Layout Parameters

- All view groups include a width and height (`layout_width` and `layout_height`), and each view is required to define them.
- You can specify width and height with exact measurements, though you probably won't want to do this often.
- More often, you will use one of these constants to set the width or height:
 - `wrap_content` tells your view to size itself to the dimensions required by its content
 - `match_parent` (replaced `fill_parent` in API Level 8) tells your view to become as big as its parent view group will allow.

Layout Parameters

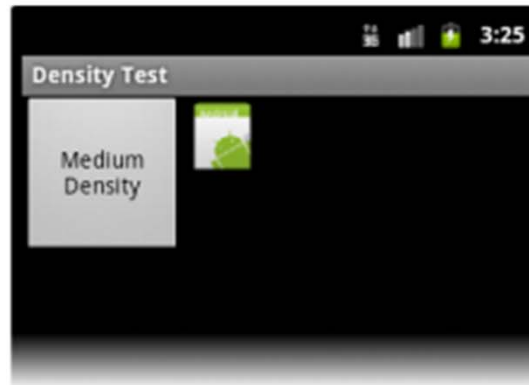
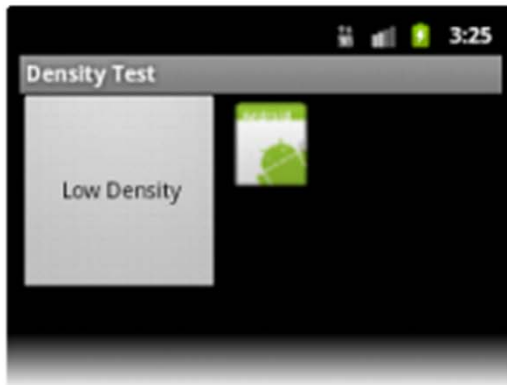
- In general, specifying a layout width and height using absolute units such as pixels is not recommended. Instead, using **relative measurements** such as:
 - **density-independent pixel** units (**dp**),
 - **wrap_content**, or
 - **match_parent**,
- is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

Screen Pixel Density

- **ldpi**: Low-density screens; approximately **120dpi**.
- **mdpi**: Medium-density (on traditional HVGA) screens; approximately **160dpi**.
- **hdpi**: High-density screens; approximately **240dpi**.
- **xhdpi**: Extra high-density screens; approximately **320dpi**.
Added in API Level 8

Screen Pixel Density

- Example application without support for different densities, as shown on low, medium, and high density screens

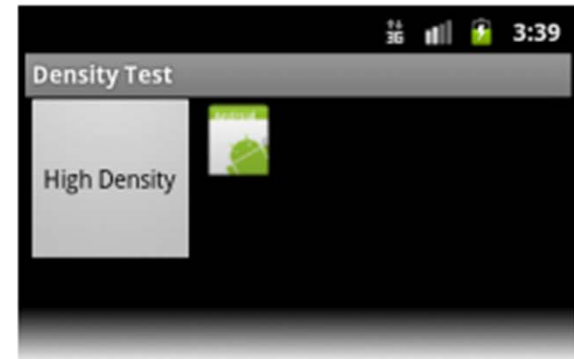
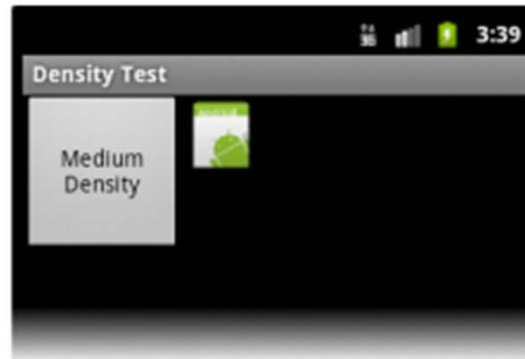
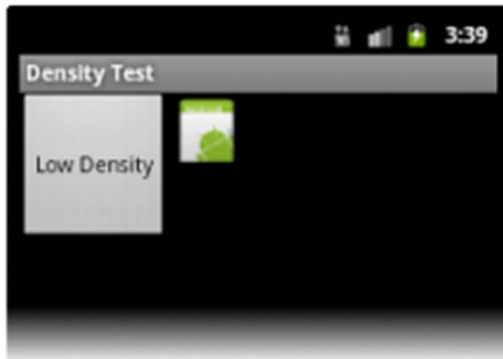


Density-independent Pixel (dp)

- A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.
- The density-independent pixel is equivalent to **one physical pixel on a 160 dpi screen**, which is the baseline density assumed by the system for a "medium" density screen.

Density-independent Pixel (dp)

- Example application with good support for different densities (it's density independent), as shown on low, medium, and high density screens.



Density-independent Pixel (dp)

| | mdpi | hdpi |
|-----------------------------------|------------|------------|
| Physical Width | 1.5 inches | 1.5 inches |
| Dots Per Inch (“dpi”) | 160 | 240 |
| Actual Pixels (Width) | 240 | 360 |
| Density (factor of baseline 160) | 1.0 | 1.5 |
| Density-independent Pixels (“dp”) | 240 | 240 |

Layout Parameters

- In general, specifying a layout width and height using absolute units such as pixels is not recommended.
- Instead, using **relative measurements** such as:
 - **density-independent pixel** units (**dp**),
 - **wrap_content**, or
 - **match_parent**,
- is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

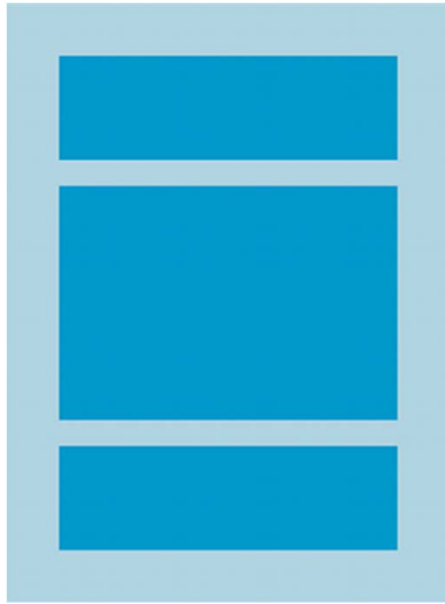
Common Layouts

- **Linear Layout**: A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.
- **Relative Layout**: Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).
- **GridLayout**: Divides its view space into rows, columns, and cells.

LINEAR LAYOUT

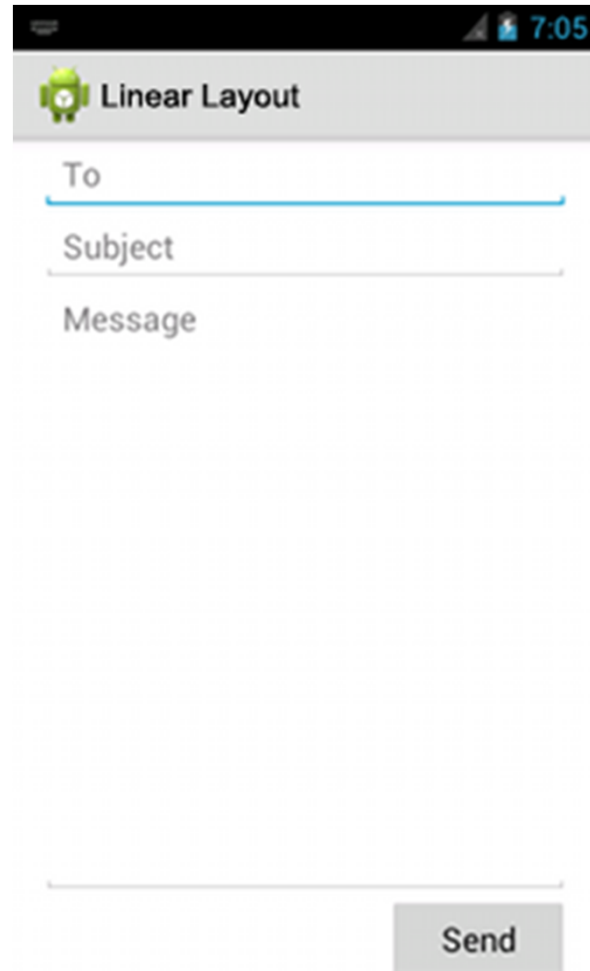
Linear Layout

- Added in API level 1, LinearLayout is a view group that aligns all children in a single direction, [vertically](#) or [horizontally](#).



- You can specify the layout direction with the [android:orientation](#) attribute.

Linear Layout



The screenshot displays an Android application window with a black status bar at the top showing the time 7:05. The app's title bar is grey and contains the Android logo and the text "Linear Layout". The main content area is white and features three vertically stacked input fields: "To" (with a blue underline), "Subject" (with a grey underline), and "Message" (a larger text area). A grey "Send" button is positioned at the bottom right of the form.

Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=http://schemas.android.com/apk/res/android
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >

</LinearLayout>
```

Linear Layout

- All children of a LinearLayout are stacked one after the other.
 - A vertical list will only have one child per row, no matter how wide they are.
 - A horizontal list will only be one row high (the height of the tallest child, plus padding).
- A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.

Linear Layout

- LinearLayout also supports assigning a **weight** to individual children with the **android:layout_weight** attribute.
- This attribute indicates how much of the **extra space** in the LinearLayout will be allocated to the view.
- Child views can specify a weight value, and then **any remaining space in the view group is assigned to children in the proportion of their declared weight**.
- Default weight is zero.

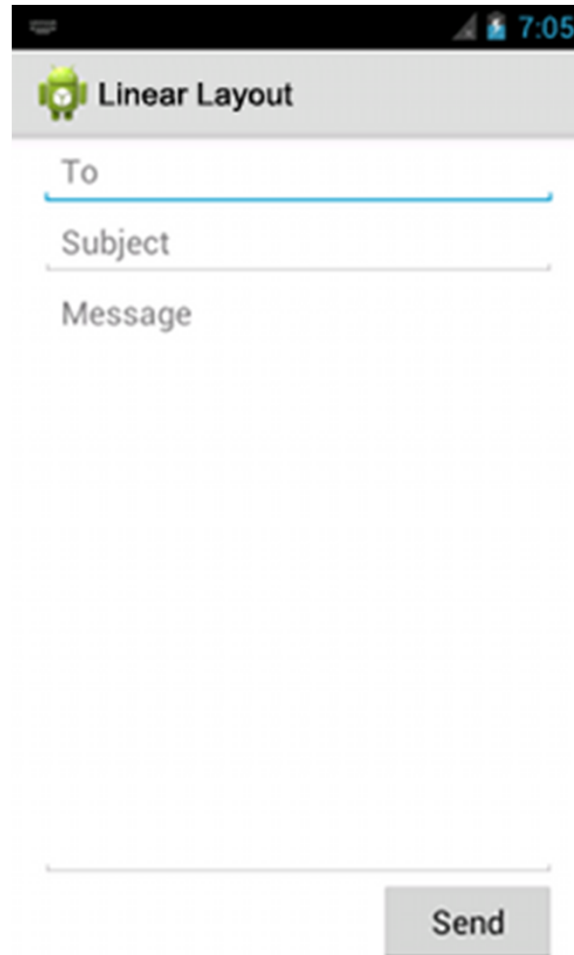
Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
</LinearLayout>
```

Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    ...
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```


Linear Layout



The image shows a screenshot of an Android application window. The title bar at the top is black and contains the text "Linear Layout" in white. Below the title bar is a light gray header area with a green Android robot icon on the left and the text "Linear Layout" in black. The main content area is white and contains three text input fields stacked vertically. The first field is labeled "To" and has a blue underline. The second field is labeled "Subject" and has a gray underline. The third field is labeled "Message" and has a gray underline. At the bottom right of the form is a gray button with the text "Send" in black.

RELATIVE LAYOUT

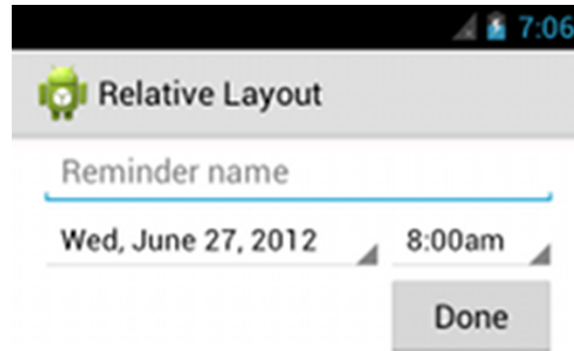
Relative Layout

- Added in API level 1, RelativeLayout is a view group that displays child views in relative positions.



- The position of each view can be specified as **relative to sibling elements** (left-of or below another view) or in positions **relative to the parent** (such as aligned to the bottom, left of center).

Relative Layout



Relative Layout

```
<?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="match_parent"  
    android:paddingLeft="16dp"  
    android:paddingRight="16dp" >  
  
</RelativeLayout>
```

Relative Layout

- A RelativeLayout is a very powerful utility for designing a user interface because it can **eliminate nested view groups** and keep your layout hierarchy flat, which **improves performance**.

Relative Layout

- RelativeLayout lets child views specify their position relative to the parent view or to each other (**specified by ID**).
- By default, all child views are drawn at the **top-left of the layout**, so you must define the position of each view using the various layout properties available from Relative Layout's Layout Parameters.

Layout Parameters

- Some of the many layout properties available to views:
 - `android:layout_alignParentTop`
If "true", makes the top edge of this view match the top edge of the parent.
 - `android:layout_centerVertical`
If "true", centers this child vertically within its parent.
 - `android:layout_below`
Positions the top edge of this view below the view specified with a resource ID.
 - `android:layout_toRightOf`
Positions the left edge of this view to the right of the view specified with a resource ID.

Layout Parameters

- The value for each layout property is either a **boolean** to enable a layout position relative to the parent or **an ID that references another view** in the layout against which the view should be positioned.
- In your XML layout, **dependencies against other views in the layout can be declared in any order.**
- For example, you can declare that "view1" be positioned below "view2" even if "view2" is the last view declared in the hierarchy.

Positions Relative to Parent

- `android:layout_alignParentTop`: Aligns view's top with the top of the parent
- `android:layout_alignParentBottom`: Aligns view's bottom with the bottom of the parent
- `android:layout_alignParentLeft`: Aligns view's left side with the left side of the parent
- `android:layout_alignParentRight`: Aligns view's right side with the right side of the parent
- `android:layout_centerHorizontal`: Positions view horizontally at the center of the parent
- `android:layout_centerVertical`: Positions view vertically at the center of the parent
- `android:layout_centerInParent`: Positions view both horizontally and vertically at the center of the parent

Positions Relative to Other Views

- `android:layout_above`: Indicates that view should be placed above the view referenced in the property
- `android:layout_below`: Indicates that view should be placed below the view referenced in the property
- `android:layout_toLeftOf`: Indicates that view should be placed to the left of the view referenced in the property
- `android:layout_toRightOf`: Indicates that view should be placed to the right of the view referenced in the property

Positions Relative to Other Views

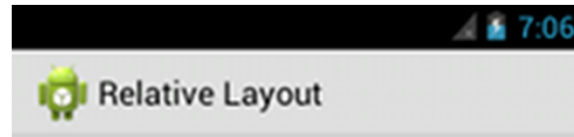
- `android:layout_alignTop`: Indicates that view's top edge should be aligned with the top edge of the view referenced in the property
- `android:layout_alignBottom`: Indicates that view's bottom edge should be aligned with the bottom edge of the view referenced in the property
- `android:layout_alignLeft`: Indicates that view's left edge should be aligned with the left edge of the view referenced in the property
- `android:layout_alignRight`: Indicates that view's right edge should be aligned with the right edge of the view referenced in the property

Relative Layout

```
<?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="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

</RelativeLayout>
```

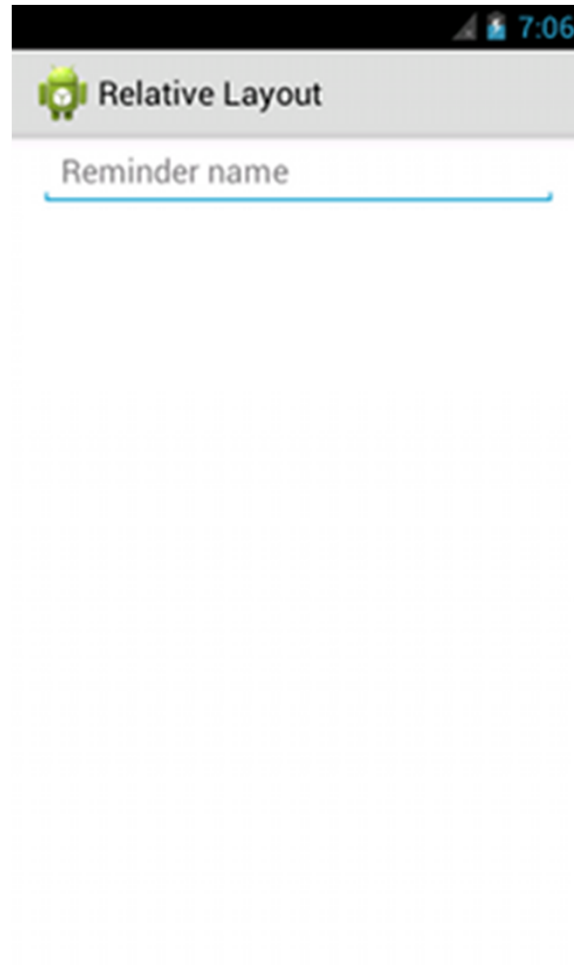
Relative Layout



Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout ... >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
</RelativeLayout>
```

Relative Layout

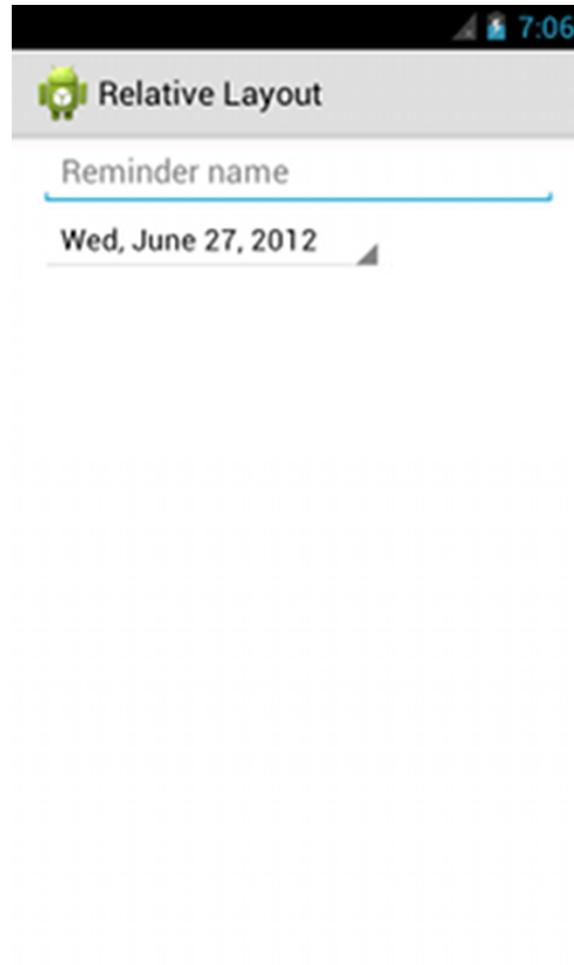


Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout ... >
    ...
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />

</RelativeLayout>
```

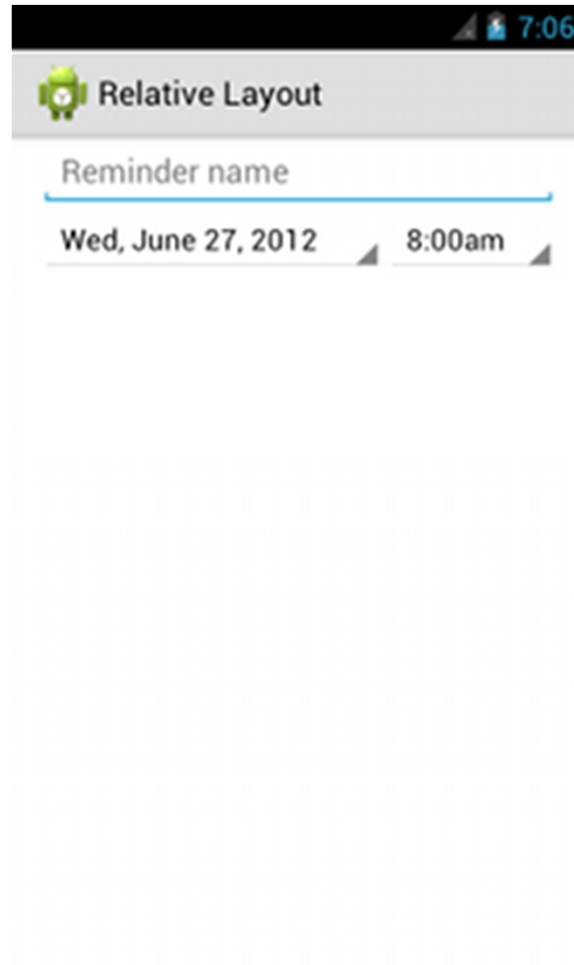
Relative Layout



Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout ... >
    ...
    ...
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
</RelativeLayout>
```

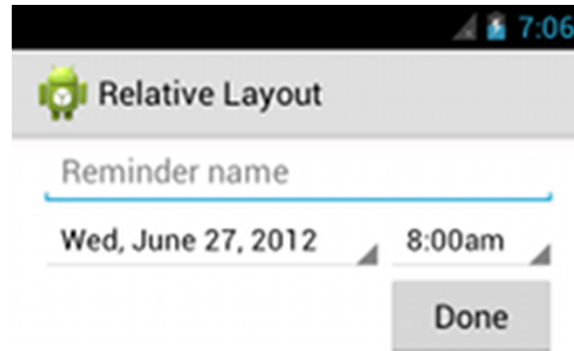
Relative Layout



Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout ... >
    ...
    ...
    ...
    ...
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```

Relative Layout



GRID LAYOUT

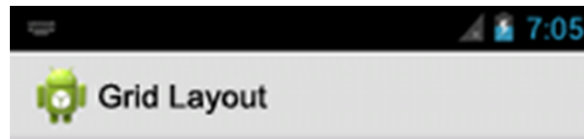
Grid Layout

- Added in API level 14, GridLayout is a layout that divides its view space into rows, columns, and cells.
- GridLayout places views in it automatically, but it is also possible to define the column and row index to place a view into GridLayout.
- With the span property of cells, it is possible to make a view span multiple rows or columns.
- GridLayout has been available since API Level 14, so the minimum SDK property should be set to 14 or greater

Grid Layout

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="3"
    android:rowCount="3"
    android:orientation="horizontal" >
    <TextView android:text="A" />
    <TextView android:text="B" />
    <TextView android:text="C" />
    <TextView android:text="D" />
    <TextView android:text="E" />
    <TextView android:text="F" />
    <TextView android:text="G" />
    <TextView android:text="H" />
    <TextView android:text="I" />
</GridLayout>
```

Grid Layout

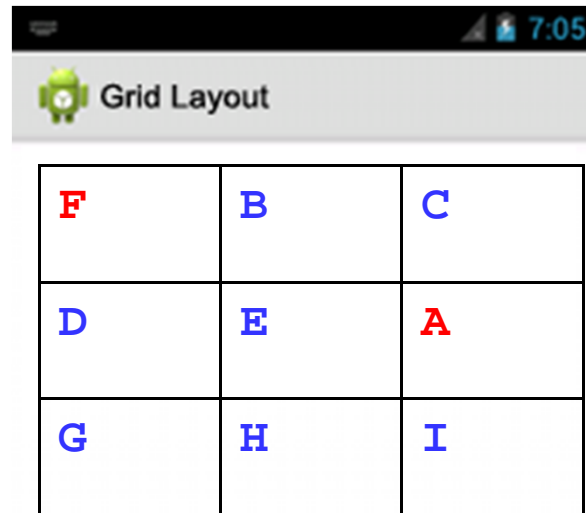


| | | |
|---|---|---|
| A | B | C |
| D | E | F |
| G | H | I |

Grid Layout

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ... >
    <TextView android:text="A"
        android:layout_row=1
        android:layout_col=2 />
    <TextView android:text="B" />
    <TextView android:text="C" />
    <TextView android:text="D" />
    <TextView android:text="E" />
    <TextView android:text="F"
        android:layout_row=0
        android:layout_col=0 />
    <TextView android:text="G" />
    <TextView android:text="H" />
    <TextView android:text="I" />
</GridLayout>
```

Grid Layout



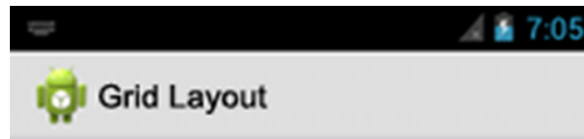
A screenshot of an Android application interface. At the top is a black status bar with a signal strength icon, a battery icon, and the time '7:05'. Below this is a grey title bar with the Android logo and the text 'Grid Layout'. The main content area displays a 3x3 grid of white squares with black borders. Each square contains a single letter. The letters are arranged as follows: Row 1: F (red), B (blue), C (blue); Row 2: D (blue), E (blue), A (red); Row 3: G (blue), H (blue), I (blue).

| | | |
|---|---|---|
| F | B | C |
| D | E | A |
| G | H | I |

Grid Layout

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ... >
    <TextView android:text="A" />
    <TextView android:text="B"
        android:layout_rowSpan="2"
        android:layout_gravity="fill"
        android:gravity="center" />
    <TextView android:text="C" />
    <TextView android:text="D" />
    <TextView android:text="E"
        android:layout_colSpan="2"
        android:layout_gravity="fill" />
    <TextView android:text="F" />
    <TextView android:text="G" />
</GridLayout>
```

Grid Layout

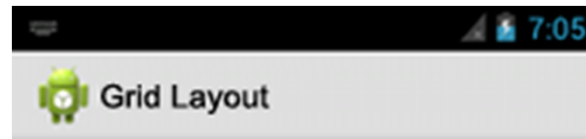


| | | |
|---|---|---|
| A | B | |
| C | D | E |
| F | G | E |

Grid Layout

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ... >
    <TextView android:text="A" />
    <TextView android:text="B" />
    <TextView android:text="C" />
    <TextView android:text="D" />
    <space
        android:layout_row=1
        android:layout_col=1
        android:layout_colSpan="2"
        android:layout_gravity="fill" />
    <TextView android:text="E" />
    <TextView android:text="F" />
    <TextView android:text="G" />
</GridLayout>
```

Grid Layout



| | | |
|---|---|---|
| A | B | C |
| D | | |
| E | F | G |

References

- Android Developers: User Interface Guide
 - <http://developer.android.com/guide/topics/ui/index.html>

Q & A