**View**

[Kotlin](https://developer.android.com/reference/kotlin/android/view/View)

public class View  
extends [Object](https://developer.android.com/reference/java/lang/Object) implements [Drawable.Callback](https://developer.android.com/reference/android/graphics/drawable/Drawable.Callback), [KeyEvent.Callback](https://developer.android.com/reference/android/view/KeyEvent.Callback), [AccessibilityEventSource](https://developer.android.com/reference/android/view/accessibility/AccessibilityEventSource)

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.view.View |
| Known direct subclasses  [AnalogClock](https://developer.android.com/reference/android/widget/AnalogClock), [ImageView](https://developer.android.com/reference/android/widget/ImageView), [KeyboardView](https://developer.android.com/reference/android/inputmethodservice/KeyboardView), [MediaRouteButton](https://developer.android.com/reference/android/app/MediaRouteButton), [ProgressBar](https://developer.android.com/reference/android/widget/ProgressBar), [Space](https://developer.android.com/reference/android/widget/Space), [SurfaceView](https://developer.android.com/reference/android/view/SurfaceView), [TextView](https://developer.android.com/reference/android/widget/TextView), [TextureView](https://developer.android.com/reference/android/view/TextureView), [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup), [ViewStub](https://developer.android.com/reference/android/view/ViewStub) | |

|  |
| --- |
| Known indirect subclasses  [AbsListView](https://developer.android.com/reference/android/widget/AbsListView), [AbsSeekBar](https://developer.android.com/reference/android/widget/AbsSeekBar), [AbsSpinner](https://developer.android.com/reference/android/widget/AbsSpinner), [AbsoluteLayout](https://developer.android.com/reference/android/widget/AbsoluteLayout), [ActionMenuView](https://developer.android.com/reference/android/widget/ActionMenuView), [AdapterView](https://developer.android.com/reference/android/widget/AdapterView)<T extends [Adapter](https://developer.android.com/reference/android/widget/Adapter)>, [AdapterViewAnimator](https://developer.android.com/reference/android/widget/AdapterViewAnimator), [AdapterViewFlipper](https://developer.android.com/reference/android/widget/AdapterViewFlipper), [AppWidgetHostView](https://developer.android.com/reference/android/appwidget/AppWidgetHostView), [AutoCompleteTextView](https://developer.android.com/reference/android/widget/AutoCompleteTextView), [Button](https://developer.android.com/reference/android/widget/Button), [CalendarView](https://developer.android.com/reference/android/widget/CalendarView), [CheckBox](https://developer.android.com/reference/android/widget/CheckBox), [CheckedTextView](https://developer.android.com/reference/android/widget/CheckedTextView), [Chronometer](https://developer.android.com/reference/android/widget/Chronometer), and 54 others. |

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.). The [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) subclass is the base class for *layouts*, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

### Using Views

All of the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files. There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

Once you have created a tree of views, there are typically a few types of common operations you may wish to perform:

* **Set properties:** for example setting the text of a [TextView](https://developer.android.com/reference/android/widget/TextView). The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.
* **Set focus:** The framework will handle moving focus in response to user input. To force focus to a specific view, call [requestFocus()](https://developer.android.com/reference/android/view/View" \l "requestFocus()).
* **Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using [setOnFocusChangeListener(android.view.View.OnFocusChangeListener)](https://developer.android.com/reference/android/view/View#setOnFocusChangeListener(android.view.View.OnFocusChangeListener)). Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.
* **Set visibility:** You can hide or show views using [setVisibility(int)](https://developer.android.com/reference/android/view/View" \l "setVisibility(int)).

### IDs

Views may have an integer id associated with them. These ids are typically assigned in the layout XML files, and are used to find specific views within the view tree. A common pattern is to:

* Define a Button in the layout file and assign it a unique ID.
* <Button
* android:id="@+id/my\_button"
* android:layout\_width="wrap\_content"
* android:layout\_height="wrap\_content"
* android:text="@string/my\_button\_text"/>

* From the onCreate method of an Activity, find the Button

      Button myButton = findViewById(R.id.my\_button);

View IDs need not be unique throughout the tree, but it is good practice to ensure that they are at least unique within the part of the tree you are searching.

### Position

The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of left and top coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.

It is possible to retrieve the location of a view by invoking the methods [getLeft()](https://developer.android.com/reference/android/view/View" \l "getLeft()) and [getTop()](https://developer.android.com/reference/android/view/View" \l "getTop()). The former returns the left, or X, coordinate of the rectangle representing the view. The latter returns the top, or Y, coordinate of the rectangle representing the view. These methods both return the location of the view relative to its parent. For instance, when getLeft() returns 20, that means the view is located 20 pixels to the right of the left edge of its direct parent.

In addition, several convenience methods are offered to avoid unnecessary computations, namely [getRight()](https://developer.android.com/reference/android/view/View" \l "getRight()) and [getBottom()](https://developer.android.com/reference/android/view/View" \l "getBottom()). These methods return the coordinates of the right and bottom edges of the rectangle representing the view. For instance, calling [getRight()](https://developer.android.com/reference/android/view/View" \l "getRight()) is similar to the following computation: getLeft() + getWidth() (see [Size](https://developer.android.com/reference/android/view/View#SizePaddingMargins) for more information about the width.)

### Size, padding and margins

The size of a view is expressed with a width and a height. A view actually possess two pairs of width and height values.

The first pair is known as measured width and measured height. These dimensions define how big a view wants to be within its parent (see [Layout](https://developer.android.com/reference/android/view/View#Layout) for more details.) The measured dimensions can be obtained by calling [getMeasuredWidth()](https://developer.android.com/reference/android/view/View" \l "getMeasuredWidth()) and [getMeasuredHeight()](https://developer.android.com/reference/android/view/View" \l "getMeasuredHeight()).

The second pair is simply known as width and height, or sometimes drawing width and drawing height. These dimensions define the actual size of the view on screen, at drawing time and after layout. These values may, but do not have to, be different from the measured width and height. The width and height can be obtained by calling [getWidth()](https://developer.android.com/reference/android/view/View" \l "getWidth()) and [getHeight()](https://developer.android.com/reference/android/view/View" \l "getHeight()).

To measure its dimensions, a view takes into account its padding. The padding is expressed in pixels for the left, top, right and bottom parts of the view. Padding can be used to offset the content of the view by a specific amount of pixels. For instance, a left padding of 2 will push the view's content by 2 pixels to the right of the left edge. Padding can be set using the [setPadding(int, int, int, int)](https://developer.android.com/reference/android/view/View" \l "setPadding(int,%20int,%20int,%20int)) or [setPaddingRelative(int, int, int, int)](https://developer.android.com/reference/android/view/View" \l "setPaddingRelative(int,%20int,%20int,%20int)) method and queried by calling [getPaddingLeft()](https://developer.android.com/reference/android/view/View#getPaddingLeft()), [getPaddingTop()](https://developer.android.com/reference/android/view/View#getPaddingTop()), [getPaddingRight()](https://developer.android.com/reference/android/view/View#getPaddingRight()), [getPaddingBottom()](https://developer.android.com/reference/android/view/View#getPaddingBottom()), [getPaddingStart()](https://developer.android.com/reference/android/view/View#getPaddingStart()), [getPaddingEnd()](https://developer.android.com/reference/android/view/View#getPaddingEnd()).

Even though a view can define a padding, it does not provide any support for margins. However, view groups provide such a support. Refer to [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) and [ViewGroup.MarginLayoutParams](https://developer.android.com/reference/android/view/ViewGroup.MarginLayoutParams) for further information.

### Themes

By default, Views are created using the theme of the Context object supplied to their constructor; however, a different theme may be specified by using the [android:theme](https://developer.android.com/reference/android/R.styleable" \l "View_theme) attribute in layout XML or by passing a [ContextThemeWrapper](https://developer.android.com/reference/android/view/ContextThemeWrapper) to the constructor from code.

When the [android:theme](https://developer.android.com/reference/android/R.styleable" \l "View_theme) attribute is used in XML, the specified theme is applied on top of the inflation context's theme (see [LayoutInflater](https://developer.android.com/reference/android/view/LayoutInflater)) and used for the view itself as well as any child elements.

In the following example, both views will be created using the Material dark color scheme; however, because an overlay theme is used which only defines a subset of attributes, the value of [android:colorAccent](https://developer.android.com/reference/android/R.styleable" \l "Theme_colorAccent) defined on the inflation context's theme (e.g. the Activity theme) will be preserved.

<LinearLayout

...

android:theme="@android:theme/ThemeOverlay.Material.Dark">

<View ...>

</LinearLayout>