public class

RadioButton

extends CompoundButton

java.lang.Object

↳android.view.View

↳android.widget.TextView

↳android.widget.Button

↳android.widget.CompoundButton

↳android.widget.RadioButton

Class Overview

A radio button is a two-states button that can be either checked or unchecked. When the radio button is unchecked, the user can press or click it to check it. However, contrary to a`CheckBox`, a radio button cannot be unchecked by the user once checked.

Radio buttons are normally used together in a `RadioGroup`. When several radio buttons live inside a radio group, checking one radio button unchecks all the others.

See the Form Stuff tutorial.

XML attributes

See `CompoundButton Attributes`, `Button Attributes`, `TextView Attributes`, `View Attributes`

Summary

[Expand] **Inherited XML Attributes**
▸From class android.widget.TextView
▸From class android.view.View
[Expand]

Inherited Constants

▸From class android.view.View

[Expand]
Inherited Fields

▸From class android.view.View

Public Constructors

	RadioButton(Context context)
	RadioButton(Context context, AttributeSet attrs)
	RadioButton(Context context, AttributeSet attrs, int defStyle)

Public Methods

void	onPopulateAccessibilityEvent(AccessibilityEvent event) Called from `dispatchPopulateAccessibilityEvent(AccessibilityEvent)` giving a chance to this Vie accessibility event with its text content.
void	toggle() Change the checked state of the view to the inverse of its current state If the radio button is already checked, this method will not toggle the radio button.

[Expand]
Inherited Methods

▸ From class android.widget.CompoundButton

▸ From class android.widget.TextView

▸ From class android.view.View

- From class java.lang.Object
- From interface android.graphics.drawable.Drawable.Callback
- From interface android.view.KeyEvent.Callback
- From interface android.view.ViewTreeObserver.OnPreDrawListener
- From interface android.view.accessibility.AccessibilityEventSource
- From interface android.widget.Checkable

Public Constructors

public ***RadioButton*** *(Context context)*

Since: API Level 1

public ***RadioButton*** *(Context context, AttributeSet attrs)*

Since: API Level 1

public ***RadioButton*** *(Context context, AttributeSet attrs, int defStyle)*

Since: API Level 1

Public Methods

public void ***onPopulateAccessibilityEvent*** *(AccessibilityEvent event)*

Since: API Level 14

Called from `dispatchPopulateAccessibilityEvent(AccessibilityEvent)` giving a chance to this View to populate the accessibility event with its text content. While this method is free to modify event attributes other than text content, doing so should normally be performed in `onInitializeAccessibilityEvent(AccessibilityEvent)`.

Example: Adding formatted date string to an accessibility event in addition to the text added by the super implementation:

```
 public void onPopulateAccessibilityEvent(AccessibilityEvent event) {
     super.onPopulateAccessibilityEvent(event);
     final int flags = DateUtils.FORMAT_SHOW_DATE |
DateUtils.FORMAT_SHOW_WEEKDAY;
     String selectedDateUtterance = DateUtils.formatDateTime(mContext,
```

```
            mCurrentDate.getTimeInMillis(), flags);
        event.getText().add(selectedDateUtterance);
    }
```

If an View.AccessibilityDelegate has been specified via calling setAccessibilityDelegate(AccessibilityDelegate) itsonPopulateAccessibilityEvent(View, AccessibilityEvent) is responsible for handling this call.

Note: Always call the super implementation before adding information to the event, in case the default implementation has basic information to add.

Parameters

event	The accessibility event which to populate.

public void ***toggle*** *()*

Since: API Level 1

Change the checked state of the view to the inverse of its current state

If the radio button is already checked, this method will not toggle the radio button.

public class

CheckBox

extends CompoundButton

java.lang.Object

↳android.view.View

↳android.widget.TextView

↳android.widget.Button

↳android.widget.CompoundButton

↳ android.widget.CheckBox

Class Overview

A checkbox is a specific type of two-states button that can be either checked or unchecked. A example usage of a checkbox inside your activity would be the following:

```
 public class MyActivity extends Activity {
     protected void onCreate(Bundle icicle) {
         super.onCreate(icicle);

         setContentView(R.layout.content_layout_id);

         final CheckBox checkBox = (CheckBox)
findViewById(R.id.checkbox_id);
         if (checkBox.isChecked()) {
             checkBox.setChecked(false);
         }
     }
 }
```

See the Form Stuff tutorial.

XML attributes

See CompoundButton Attributes, Button Attributes, TextView Attributes, View Attributes

Summary

[Expand]
Inherited XML Attributes

▸From class android.widget.TextView

▸From class android.view.View

[Expand]
Inherited Constants

▸From class android.view.View

[Expand]
Inherited Fields

▸From class android.view.View

Public Constructors

CheckBox(Context context)
CheckBox(Context context, AttributeSet attrs)
CheckBox(Context context, AttributeSet attrs, int defStyle)

Public Methods

void	onPopulateAccessibilityEvent(AccessibilityEvent event) Called from `dispatchPopulateAccessibilityEvent(AccessibilityEvent)` giving a chance to this Vie accessibility event with its text content.

[Expand]
Inherited Methods

▸ From class android.widget.CompoundButton

▸ From class android.widget.TextView

- From class android.view.View
- From class java.lang.Object
- From interface android.graphics.drawable.Drawable.Callback
- From interface android.view.KeyEvent.Callback
- From interface android.view.ViewTreeObserver.OnPreDrawListener
- From interface android.view.accessibility.AccessibilityEventSource
- From interface android.widget.Checkable

Public Constructors

public ***CheckBox*** *(Context context)*

Since: API Level 1

public ***CheckBox*** *(Context context, AttributeSet attrs)*

Since: API Level 1

public ***CheckBox*** *(Context context, AttributeSet attrs, int defStyle)*

Since: API Level 1

Public Methods

public void ***onPopulateAccessibilityEvent*** *(AccessibilityEvent event)*

Since: API Level 14

Called from `dispatchPopulateAccessibilityEvent(AccessibilityEvent)` giving a chance to this View to populate the accessibility event with its text content. While this method is free to modify event attributes other than text content, doing so should normally be performed in `onInitializeAccessibilityEvent(AccessibilityEvent)`.

Example: Adding formatted date string to an accessibility event in addition to the text added by the super implementation:

```
public void onPopulateAccessibilityEvent(AccessibilityEvent event) {
    super.onPopulateAccessibilityEvent(event);
```

```
      final int flags = DateUtils.FORMAT_SHOW_DATE |
DateUtils.FORMAT_SHOW_WEEKDAY;
      String selectedDateUtterance = DateUtils.formatDateTime(mContext,
          mCurrentDate.getTimeInMillis(), flags);
      event.getText().add(selectedDateUtterance);
  }
```

If an `View.AccessibilityDelegate` has been specified via calling `setAccessibilityDelegate(AccessibilityDelegate)` its`onPopulateAccessibilityEvent(View, AccessibilityEvent)` is responsible for handling this call.

GridLayout

extends ViewGroup

java.lang.Object

↳android.view.View

↳android.view.ViewGroup

↳android.widget.GridLayout

Class Overview

A layout that places its children in a rectangular *grid*.

The grid is composed of a set of infinitely thin lines that separate the viewing area into *cells*. Throughout the API, grid lines are referenced by grid *indices*. A grid with `N` columns has `N + 1` grid indices that run from `0` through `N` inclusive. Regardless of how GridLayout is configured, grid index `0` is fixed to the leading edge of the container and grid index `N` is fixed to its trailing edge (after padding is taken into account).

Row and Column Specs

Children occupy one or more contiguous cells, as defined by their `rowSpec` and `columnSpec` layout parameters. Each spec defines the set of rows or columns that are to be occupied; and how children should be aligned within the resulting group of cells. Although cells do not normally overlap in a GridLayout, GridLayout does not prevent children being defined to occupy the same cell or group of cells. In this case however, there is no guarantee that children will not themselves overlap after the layout operation completes.

Default Cell Assignment

If a child does not specify the row and column indices of the cell it wishes to occupy, GridLayout assigns cell locations automatically using its: `orientation`, `rowCount` and`columnCount` properties.

Space

Space between children may be specified either by using instances of the dedicated `Space` view or by setting the `leftMargin`, `topMargin`, `rightMargin` and `bottomMargin`layout parameters. When the `useDefaultMargins` property is set, default margins around children are automatically allocated based on the prevailing UI style guide for the platform. Each of the margins so defined may be independently overridden by an assignment to the appropriate layout parameter. Default values will generally produce a reasonable spacing between components but values may change between different releases of the platform.

Excess Space Distribution

GridLayout's distribution of excess space is based on *priority* rather than *weight*.

A child's ability to stretch is inferred from the alignment properties of its row and column groups (which are typically set by setting the `gravity` property of the child's layout parameters). If alignment was defined along a given axis then the component is taken as *flexible* in that direction. If no alignment was set, the component is instead assumed to be*inflexible*.

Multiple components in the same row or column group are considered to act in *parallel*. Such a group is flexible only if *all* of the components within it are flexible. Row and column groups that sit either side of a common boundary are instead considered to act in *series*. The composite group made of these two elements is flexible if *one* of its elements is flexible.

To make a column stretch, make sure all of the components inside it define a gravity. To prevent a column from stretching, ensure that one of the components in the column does not define a gravity.

When the principle of flexibility does not provide complete disambiguation, GridLayout's algorithms favour rows and columns that are closer to its *right* and *bottom* edges.

Limitations

GridLayout does not provide support for the principle of *weight*, as defined in `weight`. In general, it is not therefore possible to configure a GridLayout to distribute excess space in non-trivial proportions between multiple rows or columns.

Some common use-cases may nevertheless be accommodated as follows. To place equal amounts of space around a component in a cell group; use `CENTER` alignment (or `gravity`). For complete control over excess space distribution in a row or column; use a `LinearLayout` subview to hold the components in the associated cell group. When using either of these techniques, bear in mind that cell groups may be defined to overlap.

See `GridLayout.LayoutParams` for a full description of the layout parameters used by GridLayout.

Summary

Nested Classes		
class	GridLayout.Alignment	Alignments specify where a view should be placed within a cell group and what si
class	GridLayout.LayoutParams	Layout information associated with each of the children of a GridLayout.
class	GridLayout.Spec	A Spec defines the horizontal or vertical characteristics of a group of cells.

XML Attributes		
Attribute Name	*Related Method*	*Description*
android:alignmentMode	setAlignmentMode(int)	When set to alignMargins, causes alignmen

		between the outer boundary of a view, as d margins.
android:columnCount	setColumnCount(int)	The maxmimum number of columns to crea automatically positioning children.
android:columnOrderPreserved	setColumnOrderPreserved(boolean)	When set to true, forces column boundarie same order as column indices.
android:orientation	setOrientation(int)	The orientation property is not used during
android:rowCount	setRowCount(int)	The maxmimum number of rows to create v positioning children.
android:rowOrderPreserved	setRowOrderPreserved(boolean)	When set to true, forces row boundaries to order as row indices.
android:useDefaultMargins	setUseDefaultMargins(boolean)	When set to true, tells GridLayout to use de none are specified in a view's layout param

[Expand]
Inherited XML Attributes

▸From class android.view.ViewGroup

▸From class android.view.View

Constants

int	ALIGN_BOUNDS	This constant is an `alignmentMode`.
int	ALIGN_MARGINS	This constant is an `alignmentMode`.
int	HORIZONTAL	The horizontal orientation.
int	UNDEFINED	The constant used to indicate that a value is undefined.

int	VERTICAL	The vertical orientation.

[Expand]
Inherited Constants

▸From class android.view.ViewGroup

▸From class android.view.View

Fields

public static final GridLayout.Alignment	BASELINE	Indicates that a view should be aligned with the *baselines* of the other view
public static final GridLayout.Alignment	BOTTOM	Indicates that a view should be aligned with the *bottom* edges of the other group.
public static final GridLayout.Alignment	CENTER	Indicates that a view should be *centered* with the other views in its cell grou
public static final GridLayout.Alignment	FILL	Indicates that a view should expanded to fit the boundaries of its cell group
public static final GridLayout.Alignment	LEFT	Indicates that a view should be aligned with the *left* edges of the other view
public static final GridLayout.Alignment	RIGHT	Indicates that a view should be aligned with the *right* edges of the other vie
public static final GridLayout.Alignment	TOP	Indicates that a view should be aligned with the *top* edges of the other view

[Expand]
Inherited Fields

▸From class android.view.View

Public Constructors

	GridLayout(Context context, AttributeSet attrs, int defStyle)
	GridLayout(Context context, AttributeSet attrs)
	GridLayout(Context context)

Public Methods

GridLayout.LayoutParams	generateLayoutParams(AttributeSet attrs) Returns a new set of layout parameters based on the supplied attributes set.
int	getAlignmentMode() Returns the alignment mode.
int	getColumnCount() Returns the current number of columns.
int	getOrientation() Returns the current orientation.
int	getRowCount() Returns the current number of rows.
boolean	getUseDefaultMargins() Returns whether or not this GridLayout will allocate default margins when no correspond parameters are defined.
boolean	isColumnOrderPreserved() Returns whether or not column boundaries are ordered by their grid indices.
boolean	isRowOrderPreserved() Returns whether or not row boundaries are ordered by their grid indices.

void	requestLayout() Call this when something has changed which has invalidated the layout of this view.
void	setAlignmentMode(int alignmentMode) Sets the alignment mode to be used for all of the alignments between the children of this
void	setColumnCount(int columnCount) ColumnCount is used only to generate default column/column indices when they are not component's layout parameters.
void	setColumnOrderPreserved(boolean columnOrderPreserved) When this property is `true`, GridLayout is forced to place the column boundaries so that grid indices are in ascending order in the view.
void	setOrientation(int orientation) Orientation is used only to generate default row/column indices when they are not specif component's layout parameters.
void	setRowCount(int rowCount) RowCount is used only to generate default row/column indices when they are not specifi component's layout parameters.
void	setRowOrderPreserved(boolean rowOrderPreserved) When this property is `true`, GridLayout is forced to place the row boundaries so that the indices are in ascending order in the view.
void	setUseDefaultMargins(boolean useDefaultMargins) When `true`, GridLayout allocates default margins around children based on the child's vi

static GridLayout.Spec	spec(int start, GridLayout.Alignment alignment)
	
	Return a Spec, `spec`, where:
	
	`spec.span = [start, start + 1]`
	`spec.alignment = alignment`

static GridLayout.Spec	spec(int start, int size) Return a Spec, `spec`, where: `spec.span = [start, start + size]`

static GridLayout.Spec	spec(int start, int size, GridLayout.Alignment alignment)
	Return a Spec, `spec`, where:
	`spec.span = [start, start + size]`
	`spec.alignment = alignment`

static GridLayout.Spec	spec(int start) Return a Spec, `spec`, where: `spec.span = [start, start + 1]`
Protected Methods	
GridLayout.LayoutParams	generateDefaultLayoutParams() Returns a set of default layout parameters.
GridLayout.LayoutParams	generateLayoutParams(ViewGroup.LayoutParams p) Returns a safe set of layout parameters based on the supplied layout params.
void	onDraw(Canvas canvas) Implement this to do your drawing.
void	onLayout(boolean changed, int left, int top, int right, int bottom) Called from layout when this view should assign a size and position to each of its children
void	onMeasure(int widthSpec, int heightSpec) Measure the view and its content to determine the measured width and the measured heigh
[Expand] **Inherited Methods**	

- From class android.view.ViewGroup
- From class android.view.View
- From class java.lang.Object
- From interface android.graphics.drawable.Drawable.Callback
- From interface android.view.KeyEvent.Callback
- From interface android.view.ViewManager
- From interface android.view.ViewParent
- From interface android.view.accessibility.AccessibilityEventSource

XML Attributes

android:alignmentMode

When set to alignMargins, causes alignment to take place between the outer boundary of a view, as defined by its margins. When set to alignBounds, causes alignment to take place between the edges of the view. The default is alignMargins.
See `setAlignmentMode(int)`.

Must be one of the following constant values.

Constant	Value	Description
`alignBounds`	0	Align the bounds of the children. See `ALIGN_BOUNDS`.
`alignMargins`	1	Align the margins of the children. See `ALIGN_MARGINS`.

This corresponds to the global attribute resource symbol `alignmentMode`.

Related Methods

- setAlignmentMode(int)

android:columnCount

The maxmimum number of columns to create when automatically positioning children.

Must be an integer value, such as "100".

This may also be a reference to a resource (in the form "@[package:]type:name") or theme attribute (in the form "?[package:][type:]name") containing a value of this type.

This corresponds to the global attribute resource symbol columnCount.

Related Methods

- setColumnCount(int)

android:columnOrderPreserved

When set to true, forces column boundaries to appear in the same order as column indices. The default is true.
See setColumnOrderPreserved(boolean).

Must be a boolean value, either "true" or "false".

This may also be a reference to a resource (in the form "@[package:]type:name") or theme attribute (in the form "?[package:][type:]name") containing a value of this type.

This corresponds to the global attribute resource symbol columnOrderPreserved.

Related Methods

- setColumnOrderPreserved(boolean)

android:orientation

The orientation property is not used during layout. It is only used to allocate row and column parameters when they are not specified by its children's layout paramters. GridLayout works like LinearLayout in this case; putting all the components either in a single row or in a single column - depending on the value of this flag. In the horizontal case, a columnCount property may be additionally supplied to force new rows to be created when a row is full. The rowCount attribute may be used similarly in the vertical case. The default is horizontal.

Must be one of the following constant values.

Constant	Value	Description
horizontal	0	Defines an horizontal widget.
vertical	1	Defines a vertical widget.

This corresponds to the global attribute resource symbol orientation.

Related Methods

- setOrientation(int)

android:rowCount

The maxmimum number of rows to create when automatically positioning children.

Must be an integer value, such as "100".

This may also be a reference to a resource (in the form "@[*package*:]*type*:*name*") or theme attribute (in the form "?[*package*:][*type*:]*name*") containing a value of this type.

This corresponds to the global attribute resource symbol rowCount.

Related Methods

- setRowCount(int)

android:rowOrderPreserved

When set to true, forces row boundaries to appear in the same order as row indices. The default is true.
See setRowOrderPreserved(boolean).

Must be a boolean value, either "true" or "false".

This may also be a reference to a resource (in the form "@[*package*:]*type*:*name*") or theme attribute (in the form "?[*package*:][*type*:]*name*") containing a value of this type.

This corresponds to the global attribute resource symbol rowOrderPreserved.

Related Methods

- setRowOrderPreserved(boolean)

android:useDefaultMargins

When set to true, tells GridLayout to use default margins when none are specified in a view's layout parameters. The default value is false. See setUseDefaultMargins(boolean).

Must be a boolean value, either "true" or "false".

This may also be a reference to a resource (in the form "@[*package*:]*type*:*name*") or theme attribute (in the form "?[*package*:][*type*:]*name*") containing a value of this type.

This corresponds to the global attribute resource symbol useDefaultMargins.

Related Methods

- setUseDefaultMargins(boolean)

Constants

public static final int ***ALIGN_BOUNDS***

Since: API Level 14

This constant is an alignmentMode. When the alignmentMode is set to ALIGN_BOUNDS, alignment is made between the edges of each component's raw view boundary: i.e. the area delimited by the component's: top, left, bottom and right properties.

For example, when GridLayout is in ALIGN_BOUNDS mode, children that belong to a row group that uses TOP alignment will all return the same value when their getTop() method is called.

See Also

- setAlignmentMode(int)

Constant Value: 0 (0x00000000)

public static final int ***ALIGN_MARGINS***

Since: API Level 14

This constant is an alignmentMode. When the alignmentMode is set to ALIGN_MARGINS, the bounds of each view are extended outwards, according to their margins, before the edges of the resulting rectangle are aligned.

For example, when GridLayout is in ALIGN_MARGINS mode, the quantity top - layoutParams.topMargin is the same for all children that belong to a row group that usesTOP alignment.

See Also

- setAlignmentMode(int)

Constant Value: 1 (0x00000001)

public static final int ***HORIZONTAL***

Since: API Level 14

The horizontal orientation.

Constant Value: 0 (0x00000000)

public static final int ***UNDEFINED***

Since: API Level 14

The constant used to indicate that a value is undefined. Fields can use this value to indicate that their values have not yet been set. Similarly, methods can return this value to indicate that there is no suitable value that the implementation can return. The value used for the constant (currently MIN_VALUE) is intended to avoid confusion between valid values whose sign may not be known.

Constant Value: -2147483648 (0x80000000)

public static final int ***VERTICAL***

Since: API Level 14

The vertical orientation.

Constant Value: 1 (0x00000001)

Fields

public static final GridLayout.Alignment ***BASELINE***

Since: API Level 14

Indicates that a view should be aligned with the *baselines* of the other views in its cell group. This constant may only be used as an alignment in `rowSpecs`.

See Also

- `getBaseline()`

public static final GridLayout.Alignment ***BOTTOM***

Since: API Level 14

Indicates that a view should be aligned with the *bottom* edges of the other views in its cell group.

public static final GridLayout.Alignment ***CENTER***

Since: API Level 14

Indicates that a view should be *centered* with the other views in its cell group. This constant may be used in both `rowSpecs` and `columnSpecs`.

public static final GridLayout.Alignment ***FILL***

Since: API Level 14

Indicates that a view should expanded to fit the boundaries of its cell group. This constant may be used in both `rowSpecs` and `columnSpecs`.

public static final GridLayout.Alignment ***LEFT***

Since: API Level 14

Indicates that a view should be aligned with the *left* edges of the other views in its cell group.

public static final GridLayout.Alignment ***RIGHT***

Since: API Level 14

Indicates that a view should be aligned with the *right* edges of the other views in its cell group.

public static final GridLayout.Alignment ***TOP***

Since: API Level 14

Indicates that a view should be aligned with the *top* edges of the other views in its cell group.

Public Constructors

public ***GridLayout*** *(Context context, AttributeSet attrs, int defStyle)*

Since: API Level 14

public ***GridLayout*** *(Context context, AttributeSet attrs)*

Since: API Level 14

public ***GridLayout*** *(Context context)*

Since: API Level 14

Public Methods

public GridLayout.LayoutParams ***generateLayoutParams*** *(AttributeSet attrs)*

Since: API Level 14

Returns a new set of layout parameters based on the supplied attributes set.

Parameters

attrs the attributes to build the layout parameters from

Returns

- an instance of `ViewGroup.LayoutParams` or one of its descendants

public int ***getAlignmentMode*** *()*

Since: API Level 14

Returns the alignment mode.

Related XML Attributes

- android:alignmentMode

Returns

- the alignment mode; either `ALIGN_BOUNDS` or `ALIGN_MARGINS`

See Also

- `ALIGN_BOUNDS`
- `ALIGN_MARGINS`
- `setAlignmentMode(int)`

ProgressBar

extends View

java.lang.Object

↳android.view.View

↳android.widget.ProgressBar

▸Known Direct Subclasses
AbsSeekBar

▸Known Indirect Subclasses
RatingBar, SeekBar

Class Overview

Visual indicator of progress in some operation. Displays a bar to the user representing how far the operation has progressed; the application can change the amount of progress (modifying the length of the bar) as it moves forward. There is also a secondary progress displayable on a progress bar which is useful for displaying intermediate progress, such as the buffer level during a streaming playback progress bar.

A progress bar can also be made indeterminate. In indeterminate mode, the progress bar shows a cyclic animation without an indication of progress. This mode is used by applications when the length of the task is unknown. The indeterminate progress bar can be either a spinning wheel or a horizontal bar.

The following code example shows how a progress bar can be used from a worker thread to update the user interface to notify the user of progress:

```
public class MyActivity extends Activity {
    private static final int PROGRESS = 0x1;

    private ProgressBar mProgress;
    private int mProgressStatus = 0;

    private Handler mHandler = new Handler();

    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
```

```
        setContentView(R.layout.progressbar_activity);

        mProgress = (ProgressBar) findViewById(R.id.progress_bar);

        // Start lengthy operation in a background thread
        new Thread(new Runnable() {
            public void run() {
                while (mProgressStatus < 100) {
                    mProgressStatus = doWork();

                    // Update the progress bar
                    mHandler.post(new Runnable() {
                        public void run() {
                            mProgress.setProgress(mProgressStatus);
                        }
                    });
                }
            }
        }).start();
    }
}
```

To add a progress bar to a layout file, you can use the `<ProgressBar>` element. By default, the progress bar is a spinning wheel (an indeterminate indicator). To change to a horizontal progress bar, apply the `Widget.ProgressBar.Horizontal` style, like so:

```
<ProgressBar
    style="@android:style/Widget.ProgressBar.Horizontal"
    ... />
```

If you will use the progress bar to show real progress, you must use the horizontal bar. You can then increment the progress with `incrementProgressBy()` or `setProgress()`. By default, the progress bar is full when it reaches 100. If necessary, you can adjust the maximum value (the value for a full bar) using the `android:max` attribute. Other attributes available are listed below.

Another common style to apply to the progress bar is `Widget.ProgressBar.Small`, which shows a smaller version of the spinning wheel—useful when waiting for content to load. For example, you can insert this kind of progress bar into your default layout for a view that will be populated by some content fetched from the Internet—the spinning wheel appears immediately and when your application receives the content, it replaces the progress bar with the loaded content. For example:

```
<LinearLayout
    android:orientation="horizontal"
    ... >
    <ProgressBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="@android:style/Widget.ProgressBar.Small"
        android:layout_marginRight="5dp" />
    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/loading" />
</LinearLayout>
```

Other progress bar styles provided by the system include:

- Widget.ProgressBar.Horizontal
- Widget.ProgressBar.Small
- Widget.ProgressBar.Large
- Widget.ProgressBar.Inverse
- Widget.ProgressBar.Small.Inverse
- Widget.ProgressBar.Large.Inverse

The "inverse" styles provide an inverse color scheme for the spinner, which may be necessary if your application uses a light colored theme (a white background).

XML attributes

See ProgressBar Attributes, View Attributes

Summary

XML Attributes		
Attribute Name	*Related Method*	*Description*
android:animationResolution		Timeout between frames of animation in milliseconds Must be an integer value, such as "100".
android:indeterminate		Allows to enable the indeterminate mode.
android:indeterminateBehavior		Defines how the indeterminate mode should behave when the progres
android:indeterminateDrawable		Drawable used for the indeterminate mode.
android:indeterminateDuration		Duration of the indeterminate animation.
android:indeterminateOnly		Restricts to ONLY indeterminate mode (state-keeping progress mode w

android:interpolator		
android:max		Defines the maximum value the progress can take.
android:maxHeight		An optional argument to supply a maximum height for this view.
android:maxWidth		An optional argument to supply a maximum width for this view.
android:minHeight		
android:minWidth		
android:progress		Defines the default progress value, between 0 and max.
android:progressDrawable		Drawable used for the progress mode.
android:secondaryProgress		Defines the secondary progress value, between 0 and max.

[Expand]
Inherited XML Attributes

▸From class android.view.View

[Expand]
Inherited Constants

▸From class android.view.View

[Expand]
Inherited Fields

▸From class android.view.View

Public Constructors

ProgressBar(Context context)

Create a new progress bar with range 0...100 and initial progress of 0.

	ProgressBar(Context context, AttributeSet attrs)
	ProgressBar(Context context, AttributeSet attrs, int defStyle)

Public Methods

Drawable	getIndeterminateDrawable() Get the drawable used to draw the progress bar in indeterminate mode.
Interpolator	getInterpolator() Gets the acceleration curve type for the indeterminate animation.
synchronized int	getMax() Return the upper limit of this progress bar's range.
synchronized int	getProgress() Get the progress bar's current level of progress.
Drawable	getProgressDrawable() Get the drawable used to draw the progress bar in progress mode.
synchronized int	getSecondaryProgress() Get the progress bar's current level of secondary progress.
synchronized final void	incrementProgressBy(int diff) Increase the progress bar's progress by the specified amount.
synchronized final void	incrementSecondaryProgressBy(int diff) Increase the progress bar's secondary progress by the specified amount.
void	invalidateDrawable(Drawable dr)

	Invalidates the specified Drawable.
synchronized boolean	isIndeterminate() Indicate whether this progress bar is in indeterminate mode.
void	jumpDrawablesToCurrentState() Call `Drawable.jumpToCurrentState()` on all Drawable objects associated with this view.
void	onInitializeAccessibilityEvent(AccessibilityEvent event) Initializes an `AccessibilityEvent` with information about this View which is the event source.
void	onRestoreInstanceState(Parcelable state) Hook allowing a view to re-apply a representation of its internal state that had previously been genera by `onSaveInstanceState()`.
Parcelable	onSaveInstanceState() Hook allowing a view to generate a representation of its internal state that can later be used to create with that same state.
void	postInvalidate() Cause an invalidate to happen on a subsequent cycle through the event loop.
synchronized void	setIndeterminate(boolean indeterminate) Change the indeterminate mode for this progress bar.
void	setIndeterminateDrawable(Drawable d) Define the drawable used to draw the progress bar in indeterminate mode.
void	setInterpolator(Context context, int resID) Sets the acceleration curve for the indeterminate animation.
void	setInterpolator(Interpolator interpolator)

	Sets the acceleration curve for the indeterminate animation.
synchronized void	setMax(int max) Set the range of the progress bar to 0...
synchronized void	setProgress(int progress) Set the current progress to the specified value.
void	setProgressDrawable(Drawable d) Define the drawable used to draw the progress bar in progress mode.
synchronized void	setSecondaryProgress(int secondaryProgress) Set the current secondary progress to the specified value.
void	setVisibility(int v) Set the enabled state of this view.
Protected Methods	
void	drawableStateChanged() This function is called whenever the state of the view changes in such a way that it impacts the state c shown.
void	onAttachedToWindow() This is called when the view is attached to a window.
void	onDetachedFromWindow() This is called when the view is detached from a window.
synchronized void	onDraw(Canvas canvas) Implement this to do your drawing.

synchronized void	onMeasure(int widthMeasureSpec, int heightMeasureSpec) Measure the view and its content to determine the measured width and the measured height.
void	onSizeChanged(int w, int h, int oldw, int oldh) This is called during layout when the size of this view has changed.
void	onVisibilityChanged(View changedView, int visibility) Called when the visibility of the view or an ancestor of the view is changed.
boolean	verifyDrawable(Drawable who) If your view subclass is displaying its own Drawable objects, it should override this function and return Drawable it is displaying.

[Expand]

Inherited Methods

- From class android.view.View
- From class java.lang.Object
- From interface android.graphics.drawable.Drawable.Callback
- From interface android.view.KeyEvent.Callback
- From interface android.view.accessibility.AccessibilityEventSource

XML Attributes

android:animationResolution

Timeout between frames of animation in milliseconds

Must be an integer value, such as "`100`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `animationResolution`.

Related Methods

android:indeterminate

Allows to enable the indeterminate mode. In this mode the progress bar plays an infinite looping animation.

Must be a boolean value, either "`true`" or "`false`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `indeterminate`.

Related Methods

android:indeterminateBehavior

Defines how the indeterminate mode should behave when the progress reaches max.

Must be one of the following constant values.

Constant	Value	Description
`repeat`	1	Progress starts over from 0.
`cycle`	2	Progress keeps the current value and goes back to 0.

This corresponds to the global attribute resource symbol `indeterminateBehavior`.

Related Methods

android:indeterminateDrawable

Drawable used for the indeterminate mode.

Must be a reference to another resource, in the form "`@[+][package:]type:name`" or to a theme attribute in the form "`?[package:][type:]name`".

This corresponds to the global attribute resource symbol `indeterminateDrawable`.

Related Methods

android:indeterminateDuration

Duration of the indeterminate animation.

Must be an integer value, such as "`100`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `indeterminateDuration`.

Related Methods

android:indeterminateOnly

Restricts to ONLY indeterminate mode (state-keeping progress mode will not work).

Must be a boolean value, either "`true`" or "`false`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `indeterminateOnly`.

Related Methods

android:interpolator

Related Methods

android:max

Defines the maximum value the progress can take.

Must be an integer value, such as "`100`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `max`.

Related Methods

android:maxHeight

An optional argument to supply a maximum height for this view. See {see android.widget.ImageView#setMaxHeight} for details.

Must be a dimension value, which is a floating point number appended with a unit such as "`14.5sp`". Available units are: px (pixels), dp (density-independent pixels), sp (scaled pixels based on preferred font size), in (inches), mm (millimeters).

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `maxHeight`.

Related Methods

android:maxWidth

An optional argument to supply a maximum width for this view. See {see android.widget.ImageView#setMaxWidth} for details.

Must be a dimension value, which is a floating point number appended with a unit such as "`14.5sp`". Available units are: px (pixels), dp (density-independent pixels), sp (scaled pixels based on preferred font size), in (inches), mm (millimeters).

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `maxWidth`.

Related Methods

android:minHeight

Related Methods

android:minWidth

Related Methods

android:progress

Defines the default progress value, between 0 and max.

Must be an integer value, such as "`100`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `progress`.

Related Methods

android:progressDrawable

Drawable used for the progress mode.

Must be a reference to another resource, in the form "`@[+][package:]type:name`" or to a theme attribute in the form "`?[package:][type:]name`".

This corresponds to the global attribute resource symbol `progressDrawable`.

Related Methods

android:secondaryProgress

Defines the secondary progress value, between 0 and max. This progress is drawn between the primary progress and the background. It can be ideal for media scenarios such as showing the buffering progress while the default progress shows the play progress.

Must be an integer value, such as "`100`".

This may also be a reference to a resource (in the form "`@[package:]type:name`") or theme attribute (in the form "`?[package:][type:]name`") containing a value of this type.

This corresponds to the global attribute resource symbol `secondaryProgress`.