

[java.lang.Object](#)
↳ [android.graphics.Canvas](#)

Class Overview

The Canvas class holds the "draw" calls. To draw something, you need 4 basic components: A Bitmap to hold the pixels, a Canvas to host the draw calls (writing into the bitmap), a drawing primitive (e.g. Rect, Path, text, Bitmap), and a paint (to describe the colors and styles for the drawing).

Developer Guides

For more information about how to use Canvas, read the [Canvas and Drawables](#) ([/guide/topics/graphics/2d-graphics.html](#)) developer guide.

Summary

Nested Classes	
enum Canvas.EdgeType	
enum Canvas.VertexMode	
Constants	
int ALL_SAVE_FLAG	restore everything when restore() is called
int CLIP_SAVE_FLAG	restore the current clip when restore() is called
int CLIP_TO_LAYER_SAVE_FLAG	clip against the layer's bounds
int FULL_COLOR_LAYER_SAVE_FLAG	the layer needs to 8-bits per color component
int HAS_ALPHA_LAYER_SAVE_FLAG	the layer needs to per-pixel alpha
int MATRIX_SAVE_FLAG	restore the current matrix when restore() is called
Public Constructors	
Canvas ()	Construct an empty raster canvas.
Canvas (Bitmap bitmap)	Construct a canvas with the specified bitmap to draw into.
Public Methods	
boolean clipPath (Path path)	Intersect the current clip with the specified path.
boolean clipPath (Path path, Region.Op op)	Modify the current clip with the specified path.
boolean clipRect (Rect rect, Region.Op op)	Modify the current clip with the specified rectangle, which is expressed in local coordinates.
boolean clipRect (RectF rect, Region.Op op)	Modify the current clip with the specified rectangle.
boolean clipRect (int left, int top, int right, int bottom)	Intersect the current clip with the specified rectangle, which is expressed in local coordinates.
boolean clipRect (float left, float top, float right, float bottom)	Intersect the current clip with the specified rectangle, which is expressed in local coordinates.
boolean clipRect (RectF rect)	Intersect the current clip with the specified rectangle, which is expressed in local coordinates.
boolean clipRect (float left, float top, float right, float bottom, Region.Op op)	Modify the current clip with the specified rectangle, which is expressed in local coordinates.
boolean clipRect (Rect rect)	Intersect the current clip with the specified rectangle, which is expressed in local coordinates.
boolean clipRegion (Region region)	Intersect the current clip with the specified region.
boolean clipRegion (Region region, Region.Op op)	Modify the current clip with the specified region.
void concat (Matrix matrix)	Preconcat the current matrix with the specified matrix.
void drawARGB (int a, int r, int g, int b)	Fill the entire canvas' bitmap (restricted to the current clip) with the specified ARGB color, using srocover porterduff mode.
void drawArc (RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)	Draw the specified arc, which will be scaled to fit inside the specified oval.
void drawBitmap (int[] colors, int offset, int stride, float x, float y, int width, int height, boolean hasAlpha, Paint paint)	Treat the specified array of colors as a bitmap, and draw it.
void drawBitmap (Bitmap bitmap, Matrix matrix, Paint paint)	Draw the bitmap using the specified matrix.
void drawBitmap (int[] colors, int offset, int stride, int x, int y, int width, int height, boolean hasAlpha, Paint paint)	Legacy version of drawBitmap(int[] colors, ...) that took ints for x,y
void drawBitmap (Bitmap bitmap, Rect src, RectF dst, Paint paint)	Draw the specified bitmap, scaling/translating automatically to fill the destination rectangle.
void drawBitmap (Bitmap bitmap, float left, float top, Paint paint)	Draw the specified bitmap, with its top/left corner at (x,y), using the specified paint, transformed by the current matrix.
void drawBitmap (Bitmap bitmap, Rect src, Rect dst, Paint paint)	Draw the specified bitmap, scaling/translating automatically to fill the destination rectangle.
void drawBitmapMesh (Bitmap bitmap, int meshWidth, int meshHeight, float[] verts, int vertOffset, int[] colors, int colorOffset, Paint paint)	Draw the bitmap through the mesh, where mesh vertices are evenly distributed across the bitmap.
void drawCircle (float cx, float cy, float radius, Paint paint)	Draw the specified circle using the specified paint.
void drawColor (int color)	Fill the entire canvas' bitmap (restricted to the current clip) with the specified color, using srocover porterduff mode.
void drawColor (int color, PorterDuff.Mode mode)	Fill the entire canvas' bitmap (restricted to the current clip) with the specified color and porter-duff xfermode.
void drawLine (float startX, float startY, float stopX, float stopY, Paint paint)	Draw a line segment with the specified start and stop x,y coordinates, using the specified paint.
void drawLines (float[] pts, Paint paint)	Draw a series of lines.
void drawLines (float[] pts, int offset, int count, Paint paint)	Draw a series of lines.
void drawOval (RectF oval, Paint paint)	Draw the specified oval using the specified paint.

```

    void drawPaint(Paint paint)
        Fill the entire canvas' bitmap (restricted to the current clip) with the specified paint.
    void drawPath(Path path, Paint paint)
        Draw the specified path using the specified paint.
    void drawPicture(Picture picture, RectF dst)
        Draw the picture, stretched to fit into the dst rectangle.
    void drawPicture(Picture picture)
        Save the canvas state, draw the picture, and restore the canvas state.
    void drawPicture(Picture picture, Rect dst)
        Draw the picture, stretched to fit into the dst rectangle.
    void drawPoint(float x, float y, Paint paint)
        Helper for drawPoints() for drawing a single point.
    void drawPoints(float[] pts, int offset, int count, Paint paint)
        Draw a series of points.
    void drawPoints(float[] pts, Paint paint)
        Helper for drawPoints() that assumes you want to draw the entire array
    void drawPosText(char[] text, int index, int count, float[] pos, Paint paint)
        Draw the text in the array, with each character's origin specified by the pos array.
    void drawPosText(String text, float[] pos, Paint paint)
        Draw the text in the array, with each character's origin specified by the pos array.
    void drawRGB(int r, int g, int b)
        Fill the entire canvas' bitmap (restricted to the current clip) with the specified RGB color, using srcover porterduff mode.
    void drawRect(float left, float top, float right, float bottom, Paint paint)
        Draw the specified Rect using the specified paint.
    void drawRect(RectF rect, Paint paint)
        Draw the specified Rect using the specified paint.
    void drawRect(Rect r, Paint paint)
        Draw the specified Rect using the specified Paint.
    void drawRoundRect(RectF rect, float rx, float ry, Paint paint)
        Draw the specified round-rect using the specified paint.
    void drawText(String text, float x, float y, Paint paint)
        Draw the text, with origin at (x,y), using the specified paint.
    void drawText(CharSequence text, int start, int end, float x, float y, Paint paint)
        Draw the specified range of text, specified by start/end, with its origin at (x,y), in the specified Paint.
    void drawText(char[] text, int index, int count, float x, float y, Paint paint)
        Draw the text, with origin at (x,y), using the specified paint.
    void drawText(String text, int start, int end, float x, float y, Paint paint)
        Draw the text, with origin at (x,y), using the specified paint.
    void drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint)
        Draw the text, with origin at (x,y), using the specified paint, along the specified path.
    void drawTextOnPath(char[] text, int index, int count, Path path, float hOffset, float vOffset, Paint paint)
        Draw the text, with origin at (x,y), using the specified paint, along the specified path.
    void drawVertices(Canvas.VertexMode mode, int vertexCount, float[] verts, int vertOffset, float[] texts, int texOffset, int[] colors, int colorOffset, short[] indices, int indexOffset, int indexCount, Paint paint)
        Draw the array of vertices, interpreted as triangles (based on mode).
    final Rect getClipBounds()
        Retrieve the bounds of the current clip (in local coordinates).
    boolean getClipBounds(Rect bounds)
        Return the bounds of the current clip (in local coordinates) in the bounds parameter, and return true if it is non-empty.
    int getDensity()
        Returns the target density of the canvas.
    DrawFilter getDrawFilter()
    int getHeight()
        Returns the height of the current drawing layer
    void getMatrix(Matrix ctm)
        Return, in ctm, the current transformation matrix.
    final Matrix getMatrix()
        Return a new matrix with a copy of the canvas' current transformation matrix.
    int getMaximumBitmapHeight()
        Returns the maximum allowed height for bitmaps drawn with this canvas.
    int getMaximumBitmapWidth()
        Returns the maximum allowed width for bitmaps drawn with this canvas.
    int getSaveCount()
        Returns the number of matrix/clip states on the Canvas' private stack.
    int getWidth()
        Returns the width of the current drawing layer
    boolean isHardwareAccelerated()
        Indicates whether this Canvas uses hardware acceleration.
    boolean isOpaque()
        Return true if the device that the current layer draws into is opaque (i.e.
    boolean quickReject(Path path, Canvas.EdgeType type)
        Return true if the specified path, after being transformed by the current matrix, would lie completely outside of the current clip.
    boolean quickReject(float left, float top, float right, float bottom, Canvas.EdgeType type)
        Return true if the specified rectangle, after being transformed by the current matrix, would lie completely outside of the current clip.
    boolean quickReject(RectF rect, Canvas.EdgeType type)
        Return true if the specified rectangle, after being transformed by the current matrix, would lie completely outside of the current clip.
    void restore()
        This call balances a previous call to save(), and is used to remove all modifications to the matrix/clip state since the last save call.
    void restoreToCount(int saveCount)
    void rotate(float degrees)
        Preconcat the current matrix with the specified rotation.
    void rotate(float degrees, float px, float py)
    final void rotate(float degrees, float px, float py)
        Preconcat the current matrix with the specified rotation.
    int save()
        Saves the current matrix and clip onto a private stack.
    int save(int saveFlags)
        Based on saveFlags, can save the current matrix and clip onto a private stack.
    int saveLayer(RectF bounds, Paint paint, int saveFlags)
        This behaves the same as save(), but in addition it allocates an offscreen bitmap.

```

```
    saveLayer(float left, float top, float right, float bottom, Paint paint, int saveFlags)
    int
        Helper version of saveLayer() that takes 4 values rather than a RectF.
    saveLayerAlpha(RectF bounds, int alpha, int saveFlags)
    int
        This behaves the same as save(), but in addition it allocates an offscreen bitmap.
    saveLayerAlpha(float left, float top, float right, float bottom, int alpha, int saveFlags)
    int
        Helper for saveLayerAlpha() that takes 4 values instead of a RectF.
    scale(float sx, float sy)
    void
        Preconcat the current matrix with the specified scale.
    scale(float sx, float sy, float px, float py)
    final void
        Preconcat the current matrix with the specified scale.
    setBitmap(Bitmap bitmap)
    void
        Specify a bitmap for the canvas to draw into.
    setDensity(int density)
    void
        Specifies the density for this Canvas' backing bitmap.

    void setDrawFilter(DrawFilter filter)
    void setMatrix(Matrix matrix)
    void
        Completely replace the current matrix with the specified matrix.
    skew(float sx, float sy)
    void
        Preconcat the current matrix with the specified skew.
    translate(float dx, float dy)
    void
        Preconcat the current matrix with the specified translation

Inherited Methods \[Expand\]
► From class java.lang.Object
```

Constants

public static final int ALL_SAVE_FLAG restore everything when restore() is called Constant Value: 31 (0x0000001f)	Added in API level 1
public static final int CLIP_SAVE_FLAG restore the current clip when restore() is called Constant Value: 2 (0x00000002)	Added in API level 1
public static final int CLIP_TO_LAYER_SAVE_FLAG clip against the layer's bounds Constant Value: 16 (0x00000010)	Added in API level 1
public static final int FULL_COLOR_LAYER_SAVE_FLAG the layer needs to 8-bits per color component Constant Value: 8 (0x00000008)	Added in API level 1
public static final int HAS_ALPHA_LAYER_SAVE_FLAG the layer needs to per-pixel alpha Constant Value: 4 (0x00000004)	Added in API level 1
public static final int MATRIX_SAVE_FLAG restore the current matrix when restore() is called Constant Value: 1 (0x00000001)	Added in API level 1

Public Constructors

public Canvas () Construct an empty raster canvas. Use setBitmap() to specify a bitmap to draw into. The initial target density is DENSITY_NONE (./reference/android/graphics/Bitmap.html#DENSITY_NONE); this will typically be replaced when a target bitmap is set for the canvas.	Added in API level 1
public Canvas (Bitmap bitmap) Construct a canvas with the specified bitmap to draw into. The bitmap must be mutable. The initial target density of the canvas is the same as the given bitmap's density. Parameters <i>bitmap</i> Specifies a mutable bitmap for the canvas to draw into.	Added in API level 1

Public Methods

public boolean clipPath (Path path) Intersect the current clip with the specified path. Parameters <i>path</i> The path to intersect with the current clip Returns true if the resulting is non-empty	Added in API level 1
public boolean clipPath (Path path, Region.Op op) Modify the current clip with the specified path. Parameters <i>path</i> The path to operate on the current clip	Added in API level 1

op How the clip is modified

Returns
true if the resulting is non-empty

public boolean **clipRect** (**Rect** rect, **Region.Op** op) Added in [API level 1](#)

Modify the current clip with the specified rectangle, which is expressed in local coordinates.

Parameters
rect The rectangle to intersect with the current clip.
op How the clip is modified

Returns
true if the resulting clip is non-empty

public boolean **clipRect** (**RectF** rect, **Region.Op** op) Added in [API level 1](#)

Modify the current clip with the specified rectangle.

Parameters
rect The rect to intersect with the current clip
op How the clip is modified

Returns
true if the resulting clip is non-empty

public boolean **clipRect** (int left, int top, int right, int bottom) Added in [API level 1](#)

Intersect the current clip with the specified rectangle, which is expressed in local coordinates.

Parameters
left The left side of the rectangle to intersect with the current clip
top The top of the rectangle to intersect with the current clip
right The right side of the rectangle to intersect with the current clip
bottom The bottom of the rectangle to intersect with the current clip

Returns
true if the resulting clip is non-empty

public boolean **clipRect** (float left, float top, float right, float bottom) Added in [API level 1](#)

Intersect the current clip with the specified rectangle, which is expressed in local coordinates.

Parameters
left The left side of the rectangle to intersect with the current clip
top The top of the rectangle to intersect with the current clip
right The right side of the rectangle to intersect with the current clip
bottom The bottom of the rectangle to intersect with the current clip

Returns
true if the resulting clip is non-empty

public boolean **clipRect** (**RectF** rect) Added in [API level 1](#)

Intersect the current clip with the specified rectangle, which is expressed in local coordinates.

Parameters
rect The rectangle to intersect with the current clip.

Returns
true if the resulting clip is non-empty

public boolean **clipRect** (float left, float top, float right, float bottom, **Region.Op** op) Added in [API level 1](#)

Modify the current clip with the specified rectangle, which is expressed in local coordinates.

Parameters
left The left side of the rectangle to intersect with the current clip
top The top of the rectangle to intersect with the current clip
right The right side of the rectangle to intersect with the current clip
bottom The bottom of the rectangle to intersect with the current clip
op How the clip is modified

Returns
true if the resulting clip is non-empty

public boolean **clipRect** (**Rect** rect) Added in [API level 1](#)

Intersect the current clip with the specified rectangle, which is expressed in local coordinates.

Parameters
rect The rectangle to intersect with the current clip.

Returns
true if the resulting clip is non-empty

public boolean **clipRegion** (**Region** region) Added in [API level 1](#)

Intersect the current clip with the specified region. Note that unlike clipRect() and clipPath() which transform their arguments by the current matrix, clipRegion() assumes its argument is already in the coordinate system of the current layer's bitmap, and so not transformation is performed.

Parameters
region The region to operate on the current clip, based on op

Returns
true if the resulting is non-empty

public boolean **clipRegion** (**Region** region, **Region.Op** op) Added in [API level 1](#)

Modify the current clip with the specified region. Note that unlike clipRect() and clipPath() which transform their arguments by the current matrix, clipRegion() assumes its argument is already in the coordinate system of the current layer's bitmap, and so not transformation is performed.

Parameters

region The region to operate on the current clip, based on op
op How the clip is modified

Returns
true if the resulting is non-empty

public void **concat** ([Matrix](#) matrix) Added in [API level 1](#)
Preconcat the current matrix with the specified matrix. If the specified matrix is null, this method does nothing.

Parameters
matrix The matrix to preconcatenate with the current matrix

public void **drawARGB** (int a, int r, int g, int b) Added in [API level 1](#)
Fill the entire canvas' bitmap (restricted to the current clip) with the specified ARGB color, using srocover porterduff mode.

Parameters
a alpha component (0..255) of the color to draw onto the canvas
r red component (0..255) of the color to draw onto the canvas
g green component (0..255) of the color to draw onto the canvas
b blue component (0..255) of the color to draw onto the canvas

public void **drawArc** ([RectF](#) oval, float startAngle, float sweepAngle, boolean useCenter, [Paint](#) paint) Added in [API level 1](#)
Draw the specified arc, which will be scaled to fit inside the specified oval.
If the start angle is negative or >= 360, the start angle is treated as start angle modulo 360.
If the sweep angle is >= 360, then the oval is drawn completely. Note that this differs slightly from SkPath::arcTo, which treats the sweep angle modulo 360. If the sweep angle is negative, the sweep angle is treated as sweep angle modulo 360
The arc is drawn clockwise. An angle of 0 degrees correspond to the geometric angle of 0 degrees (3 o'clock on a watch.)

Parameters
oval The bounds of oval used to define the shape and size of the arc
startAngle Starting angle (in degrees) where the arc begins
sweepAngle Sweep angle (in degrees) measured clockwise
useCenter If true, include the center of the oval in the arc, and close it if it is being stroked. This will draw a wedge
paint The paint used to draw the arc

public void **drawBitmap** (int[] colors, int offset, int stride, float x, float y, int width, int height, boolean hasAlpha, [Paint](#) paint) Added in [API level 3](#)
Treat the specified array of colors as a bitmap, and draw it. This gives the same result as first creating a bitmap from the array, and then drawing it, but this method avoids explicitly creating a bitmap object which can be more efficient if the colors are changing often.

Parameters
colors Array of colors representing the pixels of the bitmap
offset Offset into the array of colors for the first pixel
stride The number of colors in the array between rows (must be >= width or <= -width).
x The X coordinate for where to draw the bitmap
y The Y coordinate for where to draw the bitmap
width The width of the bitmap
height The height of the bitmap
hasAlpha True if the alpha channel of the colors contains valid values. If false, the alpha byte is ignored (assumed to be 0xFF for every pixel).
paint May be null. The paint used to draw the bitmap

public void **drawBitmap** ([Bitmap](#) bitmap, [Matrix](#) matrix, [Paint](#) paint) Added in [API level 1](#)
Draw the bitmap using the specified matrix.

Parameters
bitmap The bitmap to draw
matrix The matrix used to transform the bitmap when it is drawn
paint May be null. The paint used to draw the bitmap

public void **drawBitmap** (int[] colors, int offset, int stride, int x, int y, int width, int height, boolean hasAlpha, [Paint](#) paint) Added in [API level 1](#)
Legacy version of drawBitmap(int[] colors, ...) that took ints for x,y

public void **drawBitmap** ([Bitmap](#) bitmap, [Rect](#) src, [RectF](#) dst, [Paint](#) paint) Added in [API level 1](#)
Draw the specified bitmap, scaling/translating automatically to fill the destination rectangle. If the source rectangle is not null, it specifies the subset of the bitmap to draw.
Note: if the paint contains a maskfilter that generates a mask which extends beyond the bitmap's original width/height (e.g. BlurMaskFilter), then the bitmap will be drawn as if it were in a Shader with CLAMP mode. Thus the color outside of the original width/height will be the edge color replicated.
This function *ignores the density associated with the bitmap*. This is because the source and destination rectangle coordinate spaces are in their respective densities, so must already have the appropriate scaling factor applied.

Parameters
bitmap The bitmap to be drawn
src May be null. The subset of the bitmap to be drawn
dst The rectangle that the bitmap will be scaled/translated to fit into
paint May be null. The paint used to draw the bitmap

public void **drawBitmap** ([Bitmap](#) bitmap, float left, float top, [Paint](#) paint) Added in [API level 1](#)
Draw the specified bitmap, with its top/left corner at (x,y), using the specified paint, transformed by the current matrix.
Note: if the paint contains a maskfilter that generates a mask which extends beyond the bitmap's original width/height (e.g. BlurMaskFilter), then the bitmap will be drawn as if it were in a Shader with CLAMP mode. Thus the color outside of the original width/height will be the edge color replicated.
If the bitmap and canvas have different densities, this function will take care of automatically scaling the bitmap to draw at the same density as the canvas.

Parameters
bitmap The bitmap to be drawn
left The position of the left side of the bitmap being drawn

top The position of the top side of the bitmap being drawn

paint The paint used to draw the bitmap (may be null)

public void **drawBitmap** ([Bitmap](#) bitmap, [Rect](#) src, [Rect](#) dst, [Paint](#) paint)

Added in [API level 1](#)

Draw the specified bitmap, scaling/translating automatically to fill the destination rectangle. If the source rectangle is not null, it specifies the subset of the bitmap to draw.

Note: if the paint contains a maskfilter that generates a mask which extends beyond the bitmap's original width/height (e.g. [BlurMaskFilter](#)), then the bitmap will be drawn as if it were in a Shader with CLAMP mode. Thus the color outside of the original width/height will be the edge color replicated.

This function *ignores the density associated with the bitmap*. This is because the source and destination rectangle coordinate spaces are in their respective densities, so must already have the appropriate scaling factor applied.

Parameters

bitmap The bitmap to be drawn

src May be null. The subset of the bitmap to be drawn

dst The rectangle that the bitmap will be scaled/translated to fit into

paint May be null. The paint used to draw the bitmap

public void **drawBitmapMesh** ([Bitmap](#) bitmap, int meshWidth, int meshHeight, float[] verts, int vertOffset, int[] colors, int colorOffset, [Paint](#) paint)

Added in [API level 1](#)

Draw the bitmap through the mesh, where mesh vertices are evenly distributed across the bitmap. There are meshWidth+1 vertices across, and meshHeight+1 vertices down. The verts array is accessed in row-major order, so that the first meshWidth+1 vertices are distributed across the top of the bitmap from left to right. A more general version of this method is [drawVertices\(\)](#).

Parameters

bitmap The bitmap to draw using the mesh

meshWidth The number of columns in the mesh. Nothing is drawn if this is 0

meshHeight The number of rows in the mesh. Nothing is drawn if this is 0

verts Array of x,y pairs, specifying where the mesh should be drawn. There must be at least (meshWidth+1) * (meshHeight+1) * 2 + vertOffset values in the array

vertOffset Number of verts elements to skip before drawing

colors May be null. Specifies a color at each vertex, which is interpolated across the cell, and whose values are multiplied by the corresponding bitmap colors. If not null, there must be at least (meshWidth+1) * (meshHeight+1) + colorOffset values in the array.

colorOffset Number of color elements to skip before drawing

paint May be null. The paint used to draw the bitmap

public void **drawCircle** (float cx, float cy, float radius, [Paint](#) paint)

Added in [API level 1](#)

Draw the specified circle using the specified paint. If radius is <= 0, then nothing will be drawn. The circle will be filled or framed based on the Style in the paint.

Parameters

cx The x-coordinate of the center of the circle to be drawn

cy The y-coordinate of the center of the circle to be drawn

radius The radius of the circle to be drawn

paint The paint used to draw the circle

public void **drawColor** (int color)

Added in [API level 1](#)

Fill the entire canvas' bitmap (restricted to the current clip) with the specified color, using src-over porterduff mode.

Parameters

color the color to draw onto the canvas

public void **drawColor** (int color, [PorterDuff.Mode](#) mode)

Added in [API level 1](#)

Fill the entire canvas' bitmap (restricted to the current clip) with the specified color and porter-duff xfermode.

Parameters

color the color to draw with

mode the porter-duff mode to apply to the color

public void **drawLine** (float startX, float startY, float stopX, float stopY, [Paint](#) paint)

Added in [API level 1](#)

Draw a line segment with the specified start and stop x,y coordinates, using the specified paint.

Note that since a line is always "framed", the Style is ignored in the paint.

Degenerate lines (length is 0) will not be drawn.

Parameters

startX The x-coordinate of the start point of the line

startY The y-coordinate of the start point of the line

paint The paint used to draw the line

public void **drawLines** (float[] pts, [Paint](#) paint)

Added in [API level 1](#)

public void **drawLines** (float[] pts, int offset, int count, [Paint](#) paint)

Added in [API level 1](#)

Draw a series of lines. Each line is taken from 4 consecutive values in the pts array. Thus to draw 1 line, the array must contain at least 4 values. This is logically the same as drawing the array as follows: [drawLine](#)(pts[0], pts[1], pts[2], pts[3]) followed by [drawLine](#)(pts[4], pts[5], pts[6], pts[7]) and so on.

Parameters

pts Array of points to draw [x0 y0 x1 y1 x2 y2 ...]

offset Number of values in the array to skip before drawing.

count The number of values in the array to process, after skipping "offset" of them. Since each line uses 4 values, the number of "lines" that are drawn is really (count >> 2).

paint The paint used to draw the points

public void **drawOval** ([RectF](#) oval, [Paint](#) paint)

Added in [API level 1](#)

Draw the specified oval using the specified paint. The oval will be filled or framed based on the Style in the paint.

Parameters

oval The rectangle bounds of the oval to be drawn

public void drawPaint (Paint paint) Fill the entire canvas' bitmap (restricted to the current clip) with the specified paint. This is equivalent (but faster) to drawing an infinitely large rectangle with the specified paint. Parameters <i>paint</i> The paint used to draw onto the canvas	Added in API level 1
public void drawPath (Path path, Paint paint) Draw the specified path using the specified paint. The path will be filled or framed based on the Style in the paint. Parameters <i>path</i> The path to be drawn <i>paint</i> The paint used to draw the path	Added in API level 1
public void drawPicture (Picture picture, RectF dst) Draw the picture, stretched to fit into the dst rectangle.	Added in API level 1
public void drawPicture (Picture picture) Save the canvas state, draw the picture, and restore the canvas state. This differs from <code>picture.draw(canvas)</code> , which does not perform any save/restore. Note: This forces the picture to internally call <code>endRecording()</code> (//reference/android/graphics/Picture.html#endRecording()) in order to prepare for playback. Parameters <i>picture</i> The picture to be drawn	Added in API level 1
public void drawPicture (Picture picture, Rect dst) Draw the picture, stretched to fit into the dst rectangle.	Added in API level 1
public void drawPoint (float x, float y, Paint paint) Helper for <code>drawPoints()</code> for drawing a single point.	Added in API level 1
public void drawPoints (float[] pts, int offset, int count, Paint paint) Draw a series of points. Each point is centered at the coordinate specified by <code>pts[]</code> , and its diameter is specified by the paint's stroke width (as transformed by the canvas' CTM), with special treatment for a stroke width of 0, which always draws exactly 1 pixel (or at most 4 if antialiasing is enabled). The shape of the point is controlled by the paint's Cap type. The shape is a square, unless the cap type is Round, in which case the shape is a circle. Parameters <i>pts</i> Array of points to draw [x0 y0 x1 y1 x2 y2 ...] <i>offset</i> Number of values to skip before starting to draw. <i>count</i> The number of values to process, after skipping offset of them. Since one point uses two values, the number of "points" that are drawn is really (count >> 1). <i>paint</i> The paint used to draw the points	Added in API level 1
public void drawPoints (float[] pts, Paint paint) Helper for <code>drawPoints()</code> that assumes you want to draw the entire array	Added in API level 1
public void drawPosText (char[] text, int index, int count, float[] pos, Paint paint) Draw the text in the array, with each character's origin specified by the pos array. This method does not support glyph composition and decomposition and should therefore not be used to render complex scripts. Parameters <i>text</i> The text to be drawn <i>index</i> The index of the first character to draw <i>count</i> The number of characters to draw, starting from index. <i>pos</i> Array of [x,y] positions, used to position each character <i>paint</i> The paint used for the text (e.g. color, size, style)	Added in API level 1
public void drawPosText (String text, float[] pos, Paint paint) Draw the text in the array, with each character's origin specified by the pos array. This method does not support glyph composition and decomposition and should therefore not be used to render complex scripts. Parameters <i>text</i> The text to be drawn <i>pos</i> Array of [x,y] positions, used to position each character <i>paint</i> The paint used for the text (e.g. color, size, style)	Added in API level 1
public void drawRGB (int r, int g, int b) Fill the entire canvas' bitmap (restricted to the current clip) with the specified RGB color, using sroover porterduff mode. Parameters <i>r</i> red component (0..255) of the color to draw onto the canvas <i>g</i> green component (0..255) of the color to draw onto the canvas <i>b</i> blue component (0..255) of the color to draw onto the canvas	Added in API level 1
public void drawRect (float left, float top, float right, float bottom, Paint paint) Draw the specified Rect using the specified paint. The rectangle will be filled or framed based on the Style in the paint. Parameters <i>left</i> The left side of the rectangle to be drawn <i>top</i> The top side of the rectangle to be drawn <i>right</i> The right side of the rectangle to be drawn <i>bottom</i> The bottom side of the rectangle to be drawn <i>paint</i> The paint used to draw the rect	Added in API level 1
public void drawRect (RectF rect, Paint paint) Draw the specified Rect using the specified paint. The rectangle will be filled or framed based on the Style in the paint.	Added in API level 1

Parameters

rect The rect to be drawn

paint The paint used to draw the rect

public void **drawRect** ([Rect](#) r, [Paint](#) paint)

Added in [API level 1](#)

Draw the specified Rect using the specified Paint. The rectangle will be filled or framed based on the Style in the paint.

Parameters

r The rectangle to be drawn.

paint The paint used to draw the rectangle

public void **drawRoundRect** ([RectF](#) rect, float rx, float ry, [Paint](#) paint)

Added in [API level 1](#)

Draw the specified round-rect using the specified paint. The roundrect will be filled or framed based on the Style in the paint.

Parameters

rect The rectangular bounds of the roundRect to be drawn

rx The x-radius of the oval used to round the corners

ry The y-radius of the oval used to round the corners

paint The paint used to draw the roundRect

public void **drawText** ([String](#) text, float x, float y, [Paint](#) paint)

Added in [API level 1](#)

Draw the text, with origin at (x,y), using the specified paint. The origin is interpreted based on the Align setting in the paint.

Parameters

text The text to be drawn

x The x-coordinate of the origin of the text being drawn

y The y-coordinate of the origin of the text being drawn

paint The paint used for the text (e.g. color, size, style)

public void **drawText** ([CharSequence](#) text, int start, int end, float x, float y, [Paint](#) paint)

Added in [API level 1](#)

Draw the specified range of text, specified by start/end, with its origin at (x,y), in the specified Paint. The origin is interpreted based on the Align setting in the Paint.

Parameters

text The text to be drawn

start The index of the first character in text to draw

end (end - 1) is the index of the last character in text to draw

x The x-coordinate of origin for where to draw the text

y The y-coordinate of origin for where to draw the text

paint The paint used for the text (e.g. color, size, style)

public void **drawText** ([char\[\]](#) text, int index, int count, float x, float y, [Paint](#) paint)

Added in [API level 1](#)

Draw the text, with origin at (x,y), using the specified paint. The origin is interpreted based on the Align setting in the paint.

Parameters

text The text to be drawn

x The x-coordinate of the origin of the text being drawn

y The y-coordinate of the origin of the text being drawn

paint The paint used for the text (e.g. color, size, style)

public void **drawText** ([String](#) text, int start, int end, float x, float y, [Paint](#) paint)

Added in [API level 1](#)

Draw the text, with origin at (x,y), using the specified paint. The origin is interpreted based on the Align setting in the paint.

Parameters

text The text to be drawn

start The index of the first character in text to draw

end (end - 1) is the index of the last character in text to draw

x The x-coordinate of the origin of the text being drawn

y The y-coordinate of the origin of the text being drawn

paint The paint used for the text (e.g. color, size, style)

public void **drawTextOnPath** ([String](#) text, [Path](#) path, float hOffset, float vOffset, [Paint](#) paint)

Added in [API level 1](#)

Draw the text, with origin at (x,y), using the specified paint, along the specified path. The paint's Align setting determines where along the path to start the text.

Parameters

text The text to be drawn

path The path the text should follow for its baseline

hOffset The distance along the path to add to the text's starting position

vOffset The distance above(-) or below(+) the path to position the text

paint The paint used for the text (e.g. color, size, style)

public void **drawTextOnPath** ([char\[\]](#) text, int index, int count, [Path](#) path, float hOffset, float vOffset, [Paint](#) paint)

Added in [API level 1](#)

Draw the text, with origin at (x,y), using the specified paint, along the specified path. The paint's Align setting determines where along the path to start the text.

Parameters

text The text to be drawn

path The path the text should follow for its baseline

hOffset The distance along the path to add to the text's starting position

vOffset The distance above(-) or below(+) the path to position the text

paint The paint used for the text (e.g. color, size, style)

public void **drawVertices** ([Canvas.VertexMode](#) mode, int vertexCount, float[] verts, int vertOffset, float[] texts, int texOffset, int[] colors, int colorOffset, short[] indices, int indexOffset, int indexCount, [Paint](#) paint)

Added in [API level 1](#)

Draw the array of vertices, interpreted as triangles (based on mode). The verts array is required, and specifies the x,y pairs for each vertex. If texts is non-null, then it is used to specify the coordinate in shader coordinates to use at each vertex (the paint must have a shader in this case). If there is no texts array, but there is a

color array, then each color is interpolated across its corresponding triangle in a gradient. If both texts and colors arrays are present, then they behave as before, but the resulting color at each pixels is the result of multiplying the colors from the shader and the color-gradient together. The indices array is optional, but if it is present, then it is used to specify the index of each triangle, rather than just walking through the arrays in order.

Parameters	
<i>mode</i>	How to interpret the array of vertices
<i>vertexCount</i>	The number of values in the vertices array (and corresponding texts and colors arrays if non-null). Each logical vertex is two values (x, y), vertexCount must be a multiple of 2.
<i>verts</i>	Array of vertices for the mesh
<i>vertOffset</i>	Number of values in the verts to skip before drawing.
<i>texts</i>	May be null. If not null, specifies the coordinates to sample into the current shader (e.g. bitmap tile or gradient)
<i>texOffset</i>	Number of values in texts to skip before drawing.
<i>colors</i>	May be null. If not null, specifies a color for each vertex, to be interpolated across the triangle.
<i>colorOffset</i>	Number of values in colors to skip before drawing.
<i>indices</i>	If not null, array of indices to reference into the vertex (texts, colors) array.
<i>indexCount</i>	number of entries in the indices array (if not null).
<i>paint</i>	Specifies the shader to use if the texts array is non-null.

public final [Rect](#) **getClipBounds** ()

Added in [API level 1](#)

Retrieve the bounds of the current clip (in local coordinates).

Returns

the clip bounds, or [0, 0, 0, 0] if the clip is empty.

public boolean **getClipBounds** ([Rect](#) bounds)

Added in [API level 1](#)

Return the bounds of the current clip (in local coordinates) in the bounds parameter, and return true if it is non-empty. This can be useful in a way similar to quickReject, in that it tells you that drawing outside of these bounds will be clipped out.

Parameters

bounds Return the clip bounds here. If it is null, ignore it but still return true if the current clip is non-empty.

Returns

true if the current clip is non-empty.

public int **getDensity** ()

Added in [API level 4](#)

Returns the target density of the canvas. The default density is derived from the density of its backing bitmap, or [DENSITY_NONE](#) ([/reference/android/graphics/Bitmap.html#DENSITY_NONE](#)) if there is not one.

Returns

Returns the current target density of the canvas, which is used to determine the scaling factor when drawing a bitmap into it.

See Also

[setDensity\(int\)](#)
[getDensity\(\)](#)

public [DrawFilter](#) **getDrawFilter** ()

Added in [API level 1](#)

public int **getHeight** ()

Added in [API level 1](#)

Returns the height of the current drawing layer

Returns

the height of the current drawing layer

public void **getMatrix** ([Matrix](#) ctm)

Added in [API level 1](#)

Return, in ctm, the current transformation matrix. This does not alter the matrix in the canvas, but just returns a copy of it.

public final [Matrix](#) **getMatrix** ()

Added in [API level 1](#)

Return a new matrix with a copy of the canvas' current transformation matrix.

public int **getMaximumBitmapHeight** ()

Added in [API level 14](#)

Returns the maximum allowed height for bitmaps drawn with this canvas. Attempting to draw with a bitmap taller than this value will result in an error.

See Also

[getMaximumBitmapWidth\(\)](#)

public int **getMaximumBitmapWidth** ()

Added in [API level 14](#)

Returns the maximum allowed width for bitmaps drawn with this canvas. Attempting to draw with a bitmap wider than this value will result in an error.

See Also

[getMaximumBitmapHeight\(\)](#)

public int **getSaveCount** ()

Added in [API level 1](#)

Returns the number of matrix/clip states on the Canvas' private stack. This will equal # save() calls - # restore() calls.

public int **getWidth** ()

Added in [API level 1](#)

Returns the width of the current drawing layer

Returns

the width of the current drawing layer

public boolean **isHardwareAccelerated** ()

Added in [API level 11](#)

Indicates whether this Canvas uses hardware acceleration. Note that this method does not define what type of hardware acceleration may or may not be used.

Returns

True if drawing operations are hardware accelerated, false otherwise.

public boolean **isOpaque** ()

Added in [API level 1](#)

Return true if the device that the current layer draws into is opaque (i.e. does not support per-pixel alpha).

Returns

true if the device that the current layer draws into is opaque

public boolean **quickReject** ([Path](#) path, [Canvas.EdgeType](#) type)

Added in [API level 1](#)

Return true if the specified path, after being transformed by the current matrix, would lie completely outside of the current clip. Call this to check if an area you intend to draw into is clipped out (and therefore you can skip making the draw calls). Note: for speed it may return false even if the path itself might not intersect the clip (i.e. the bounds of the path intersects, but the path does not).

Parameters

path The path to compare with the current clip
type [AA](#) if the path should be considered antialiased, since that means it may affect a larger area (more pixels) than non-antialiased ([BW](#)).

Returns

true if the path (transformed by the canvas' matrix) does not intersect with the canvas' clip

public boolean **quickReject** (float left, float top, float right, float bottom, [Canvas.EdgeType](#) type)

Added in [API level 1](#)

Return true if the specified rectangle, after being transformed by the current matrix, would lie completely outside of the current clip. Call this to check if an area you intend to draw into is clipped out (and therefore you can skip making the draw calls).

Parameters

left The left side of the rectangle to compare with the current clip
top The top of the rectangle to compare with the current clip
right The right side of the rectangle to compare with the current clip
bottom The bottom of the rectangle to compare with the current clip
type [AA](#) if the path should be considered antialiased, since that means it may affect a larger area (more pixels) than non-antialiased ([BW](#)).

Returns

true if the rect (transformed by the canvas' matrix) does not intersect with the canvas' clip

public boolean **quickReject** ([RectF](#) rect, [Canvas.EdgeType](#) type)

Added in [API level 1](#)

Return true if the specified rectangle, after being transformed by the current matrix, would lie completely outside of the current clip. Call this to check if an area you intend to draw into is clipped out (and therefore you can skip making the draw calls).

Parameters

rect the rect to compare with the current clip
type [AA](#) if the path should be considered antialiased, since that means it may affect a larger area (more pixels) than non-antialiased ([BW](#)).

Returns

true if the rect (transformed by the canvas' matrix) does not intersect with the canvas' clip

public void **restore** ()

Added in [API level 1](#)

This call balances a previous call to `save()`, and is used to remove all modifications to the matrix/clip state since the last save call. It is an error to call `restore()` more times than `save()` was called.

public void **restoreToCount** (int saveCount)

Added in [API level 1](#)

Efficient way to pop any calls to `save()` that happened after the save count reached `saveCount`. It is an error for `saveCount` to be less than 1. Example: `int count = canvas.save(); ... // more calls potentially to save() canvas.restoreToCount(count); // now the canvas is back in the same state it was before the initial // call to save()`.

Parameters

saveCount The save level to restore to.

public void **rotate** (float degrees)

Added in [API level 1](#)

Preconcat the current matrix with the specified rotation.

Parameters

degrees The amount to rotate, in degrees

public final void **rotate** (float degrees, float px, float py)

Added in [API level 1](#)

Preconcat the current matrix with the specified rotation.

Parameters

degrees The amount to rotate, in degrees
px The x-coord for the pivot point (unchanged by the rotation)
py The y-coord for the pivot point (unchanged by the rotation)

public int **save** ()

Added in [API level 1](#)

Saves the current matrix and clip onto a private stack. Subsequent calls to `translate`, `scale`, `rotate`, `skew`, `concat` or `clipRect`, `clipPath` will all operate as usual, but when the balancing call to `restore()` is made, those calls will be forgotten, and the settings that existed before the `save()` will be reinstated.

Returns

The value to pass to `restoreToCount()` to balance this `save()`

public int **save** (int saveFlags)

Added in [API level 1](#)

Based on `saveFlags`, can save the current matrix and clip onto a private stack. Subsequent calls to `translate`, `scale`, `rotate`, `skew`, `concat` or `clipRect`, `clipPath` will all operate as usual, but when the balancing call to `restore()` is made, those calls will be forgotten, and the settings that existed before the `save()` will be reinstated.

Parameters

saveFlags flag bits that specify which parts of the Canvas state to save/restore

Returns

The value to pass to `restoreToCount()` to balance this `save()`

public int **saveLayer** ([RectF](#) bounds, [Paint](#) paint, int saveFlags)

Added in [API level 1](#)

This behaves the same as `save()`, but in addition it allocates an offscreen bitmap. All drawing calls are directed there, and only when the balancing call to `restore()` is made is that offscreen transferred to the canvas (or the previous layer). Subsequent calls to `translate`, `scale`, `rotate`, `skew`, `concat` or `clipRect`, `clipPath` all operate on this copy. When the balancing call to `restore()` is made, this copy is deleted and the previous matrix/clip state is restored.

Parameters

bounds May be null. The maximum size the offscreen bitmap needs to be (in local coordinates)
paint This is copied, and is applied to the offscreen when `restore()` is called.
saveFlags see `_SAVE_FLAG` constants

Returns

value to pass to `restoreToCount()` to balance this `save()`

public int **saveLayer** (float left, float top, float right, float bottom, [Paint](#) paint, int saveFlags)

Added in [API level 1](#)

Helper version of saveLayer() that takes 4 values rather than a RectF.

public int **saveLayerAlpha** ([RectF](#) bounds, int alpha, int saveFlags)

Added in [API level 1](#)

This behaves the same as save(), but in addition it allocates an offscreen bitmap. All drawing calls are directed there, and only when the balancing call to restore() is made is that offscreen transferred to the canvas (or the previous layer). Subsequent calls to translate, scale, rotate, skew, concat or clipRect, clipPath all operate on this copy. When the balancing call to restore() is made, this copy is deleted and the previous matrix/clip state is restored.

Parameters

bounds The maximum size the offscreen bitmap needs to be (in local coordinates)
alpha The alpha to apply to the offscreen when when it is drawn during restore()
saveFlags see [_SAVE_FLAG](#) constants

Returns

value to pass to restoreToCount() to balance this call

public int **saveLayerAlpha** (float left, float top, float right, float bottom, int alpha, int saveFlags)

Added in [API level 1](#)

Helper for saveLayerAlpha() that takes 4 values instead of a RectF.

public void **scale** (float sx, float sy)

Added in [API level 1](#)

Preconcat the current matrix with the specified scale.

Parameters

sx The amount to scale in X
sy The amount to scale in Y

public final void **scale** (float sx, float sy, float px, float py)

Added in [API level 1](#)

Preconcat the current matrix with the specified scale.

Parameters

sx The amount to scale in X
sy The amount to scale in Y
px The x-coord for the pivot point (unchanged by the scale)
py The y-coord for the pivot point (unchanged by the scale)

public void **setBitmap** ([Bitmap](#) bitmap)

Added in [API level 1](#)

Specify a bitmap for the canvas to draw into. All canvas state such as layers, filters, and the save/restore stack are reset with the exception of the current matrix and clip stack. Additionally, as a side-effect the canvas' target density is updated to match that of the bitmap.

Parameters

bitmap Specifies a mutable bitmap for the canvas to draw into.

See Also

[setDensity\(int\)](#)
[getDensity\(\)](#)

public void **setDensity** (int density)

Added in [API level 4](#)

Specifies the density for this Canvas' backing bitmap. This modifies the target density of the canvas itself, as well as the density of its backing bitmap via [Bitmap.setDensity\(int\)](#) ([//reference/android/graphics/Bitmap.html#setDensity\(int\)](#)).

Parameters

density The new target density of the canvas, which is used to determine the scaling factor when drawing a bitmap into it. Use [DENSITY_NONE](#) to disable bitmap scaling.

See Also

[getDensity\(\)](#)
[setDensity\(int\)](#)

public void **setDrawFilter** ([DrawFilter](#) filter)

Added in [API level 1](#)

public void **setMatrix** ([Matrix](#) matrix)

Added in [API level 1](#)

Completely replace the current matrix with the specified matrix. If the matrix parameter is null, then the current matrix is reset to identity. **Note:** It is recommended to use [concat\(Matrix\)](#) ([//reference/android/graphics/Canvas.html#concat\(android.graphics.Matrix\)](#)), [scale\(float, float\)](#) ([//reference/android/graphics/Canvas.html#scale\(float, float\)](#)), [translate\(float, float\)](#) ([//reference/android/graphics/Canvas.html#translate\(float, float\)](#)) and [rotate\(float\)](#) ([//reference/android/graphics/Canvas.html#rotate\(float\)](#)) instead of this method.

Parameters

matrix The matrix to replace the current matrix with. If it is null, set the current matrix to identity.

See Also

[concat\(Matrix\)](#)

public void **skew** (float sx, float sy)

Added in [API level 1](#)

Preconcat the current matrix with the specified skew.

Parameters

sx The amount to skew in X
sy The amount to skew in Y

public void **translate** (float dx, float dy)

Added in [API level 1](#)

Preconcat the current matrix with the specified translation

Parameters

dx The distance to translate in X
dy The distance to translate in Y