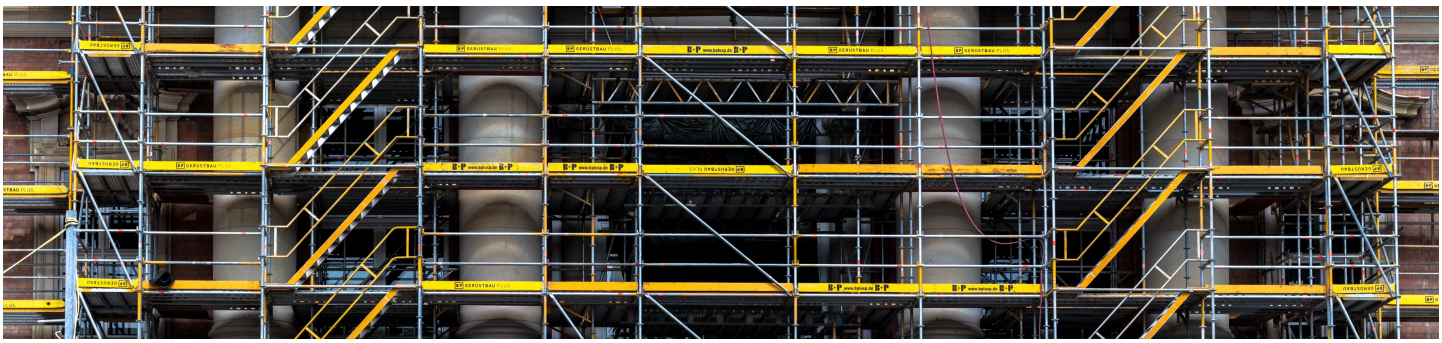


Photo by Reto Simonet on Unsplash

Drawing multiline text to Canvas on Android

Leveraging the Android framework and Kotlin to make Canvas text drawing more powerful

Nick Rout
Apr 6, 2018 · 7 min read

Featured in Android Weekly Issue #304

. . .

The Android `Canvas` offers a variety of drawing functions for implementing custom graphics in your app. A common use of `Canvas` is to draw text to a given region of a custom `View`, `Drawable`, `Bitmap`, etc.

`Canvas` has existing functions that allow you to draw text, the simplest of which can be seen below:

```
canvas.drawText(text, x, y, paint)
```

A single line of text is drawn at a given (x, y) origin, taking into account the properties of the `Paint` (to describe the colors and styles for the drawing eg. `color`, `textSize`,

The limitations of Canvas text drawing

The existing Canvas text drawing functions are simple and powerful but aren't without their limitations. The major drawback (as mentioned above) is that the text is drawn on a *single line*. If the width of the text exceeds the width of the `Canvas`, the text will be clipped. Long text will usually need to be drawn on multiple lines, and you may have wanted "paragraph" style text in the first place. From here on, we'll refer to this as multiline text.

How do we draw multiline text?

So how should we go about implementing this? Unfortunately you can't just include `\n` characters in your text, as all whitespace characters are interpreted and drawn as spaces within the single line. `Paint` includes handy `measureText` and `breakText` functions for splitting up text which you could use. You may even consider an existing algorithm such as the Knuth-Plass Line Wrapping Algorithm. This quickly becomes a complex problem.

Android framework to the rescue

Thankfully, the Android framework provides us with a class that handles all of the complexity for us: `Layout` (in the `android.text` package), described as *"a base class that manages text layout in visual elements on the screen"*. It forms the basis of how classes like `TextView` fit text within given layout parameters.

The documentation stipulates:

> *For text that will be edited, use a `DynamicLayout`, which will be updated as the text changes. For text that will not change, use a `StaticLayout`.*

Considering that we are trying to draw some static (multiline) text to Canvas, `StaticLayout` is just what we need!

Using StaticLayout

Using `StaticLayout` is quite simple. Firstly, instantiate one by obtaining and using a `StaticLayout.Builder`:

```
            .build()
```

A few parameters are required when obtaining the builder:

- `text` : The text `CharSequence` to be laid out, optionally with spans
- `start` : The index of the start of the text
- `end` : The index + 1 of the end of the text
- `textPaint` : The `TextPaint` used for layout
- `width` : The bounding width (in pixels)

These parameters allow `StaticLayout` to layout the text appropriately within the bounding `width` . A resultant `height` property becomes available once the `StaticLayout` has been instantiated. Many other parameters can be appended to the builder to adjust the end appearance, but we'll get to those later.

Note: `StaticLayout.Builder` was added in API Level 23. Prior to this, you need to use the `StaticLayout` constructors. To add to this, the constructors have been deprecated in API Level 28. Be sure to handle backwards compatibility appropriately.

Then draw!

```
staticLayout.draw(canvas)
```

Note: If you happen to be utilising this in the `onDraw` function of a custom `View` , be sure to instantiate the `StaticLayout` separately to avoid object allocation during drawing (in a constructor or Kotlin `init` block, for example).

A word on TextPaint ☝

Before we progress, there's one thing that needs to be discussed… What is `TextPaint` ? While almost all `Canvas` functions require a `Paint` parameter, `StaticLayout` requires a `TextPaint` . From the documentation, it is described as *"an extension of Paint that leaves room for some extra data used during text measuring and drawing"*. A great explanation

Exploring the StaticLayout properties 🔍

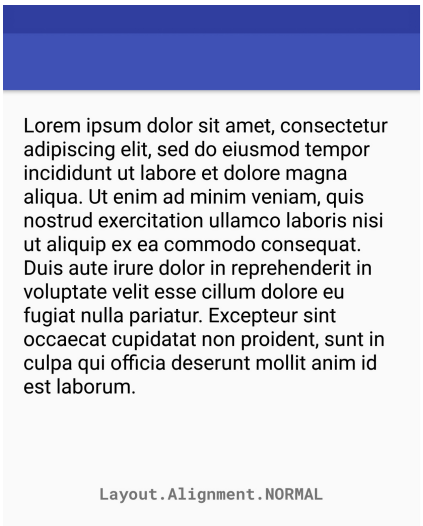
Phew! We now know the basics of what `StaticLayout` is and how we can use it to draw multiline text to `Canvas`. However, there are *a lot* of parameters that you can provide to change the appearance of the end result.

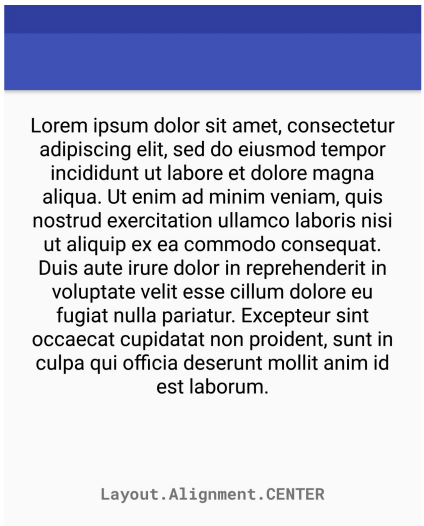
Consider a basic text editor: you usually have options to change the alignment, margins, line spacing, text size and more. The same is true for `StaticLayout`; additional parameters can be appended to the `StaticLayout.Builder`.

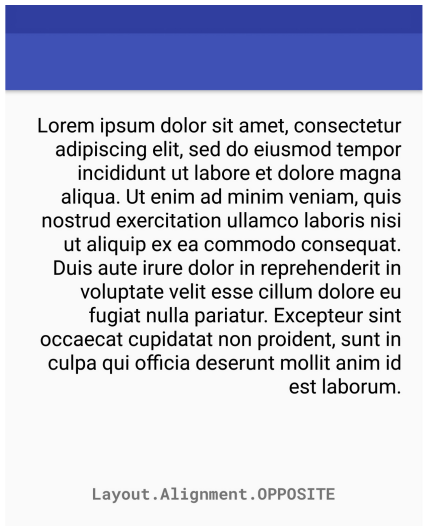
It is important to note that some parameters (like `color`) do not belong directly to `StaticLayout`, but rather belong to the `TextPaint`. Let's take a look at the some of the options we have:

alignment

The alignment of the text, similar to gravity.

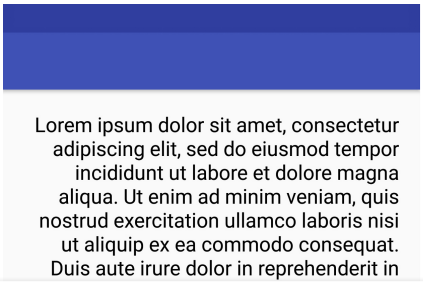


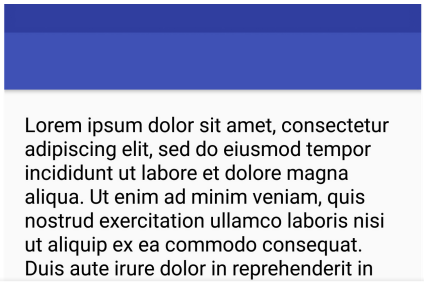




textDirection

The horizontal direction that the text follows.





TextDirectionHeuristics.RTL

TextDirectionHeuristics.LTR

lineSpacing

The spacing between lines of text (includes `spacingMult` and `spacingAdd`).

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

`spacingMult = 1.2f`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

`spacingAdd = 5f`

justificationMode (minSdk 26)

Options to justify the text (stretch spaces so that the lines appear “square”).

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

`Layout.JUSTIFICATION_MODE_NONE`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

`Layout.JUSTIFICATION_MODE_INTER_WORD`

Other properties include `ellipsize`, `maxLines`, `indents` and more. Check the `StaticLayout` and `StaticLayout.Builder` documentation for what’s available.

Positioning multiline text 🔑

we may want to position the text elsewhere (as we are able to do with the default `Canvas` text drawing methods).

We know that we define the bounding `width` of this block, and that we get a resultant `height` once we've instantiated the `StaticLayout`. We can use these properties along with a basic `Canvas` translation to position the text.

To do this, we declare a `StaticLayout` extension function. It makes use of the handy `Canvas.withTranslation` function found in the Android KTX library:

```
fun StaticLayout.draw(canvas: Canvas, x: Float, y: Float) {
    canvas.withTranslation(x, y) {
        draw(this)
    }
}
```

Using Kotlin to make it feel familiar 😁

There may be multiple places in an app in which we need to implement multiline text drawing. Instantiating a `StaticLayout` in every place would lead to unnecessary bloat. So let's make an extension function for `Canvas`!

```
fun Canvas.drawMultilineText(
        text: CharSequence,
        textPaint: TextPaint,
        width: Int,
        x: Float,
        y: Float,
        start: Int = 0,
        end: Int = text.length,
        alignment: Layout.Alignment = Layout.Alignment.ALIGN_NORMAL,
        textDir: TextDirectionHeuristic =
TextDirectionHeuristics.LTR,
        spacingMult: Float = 1f,
        spacingAdd: Float = 0f,
        hyphenationFrequency: Int =
Layout.HYPHENATION_FREQUENCY_NONE,
        justificationMode: Int = Layout.JUSTIFICATION_MODE_NONE) {

    val staticLayout = StaticLayout.Builder
            .obtain(text, start, end, textPaint, width)
            .setAlignment(alignment)
            .setTextDirection(textDir)
```

```
                .build()

        staticLayout.draw(this, x, y)
    }
```

This extension function includes most (not all) of the `StaticLayout` properties, and provides default values for those that may not be used as commonly. It also makes use of our previously defined extension function to position the block of text.

We can now draw multiline text to `Canvas` in a way that feels very familiar to the existing text drawing functions:

```
canvas.drawMultilineText(text, textPaint, width, x, y)
```

But there's one issue... 😬

Our new `drawMultilineText` extension function is great to use, but it's doing something it shouldn't: it instantiates a new `StaticLayout` every time it is called. This violates our goal to avoid object allocation during drawing.

Implementing a StaticLayout cache 💾

There's most likely a variety of ways to solve the aforementioned issue. One approach is to implement a basic `LruCache` to store/retrieve `StaticLayout`s for drawing (again making use of the Android KTX library for the `lruCache` extension function):

```
object StaticLayoutCache {

    private const val MAX_SIZE = 50 // Max number of cached items
    private val cache = lruCache<String, StaticLayout>(MAX_SIZE)

    operator fun set(key: String, staticLayout: StaticLayout) {
        cache.put(key, staticLayout)
    }

    operator fun get(key: String): StaticLayout? {
        return cache[key]
    }
}
```

```
val staticLayout = StaticLayoutCache[cacheKey] ?:
    StaticLayout.Builder.obtain(text, start, end, textPaint, width)
        ... // Add other properties
        .build().apply { StaticLayoutCache[cacheKey] = this }
```

The final piece of the puzzle is deciding what to use as the `cacheKey`. Using `StaticLayout.toString()` comes to mind, but this means we would need to instantiate it first, which we don't want to do if there's a cached version. Given that the parameters of the `drawMultilineText` function essentially describe the uniqueness of a `StaticLayout` for our purposes, we can create our own key like so:

```
val cacheKey =
"$text-$start-$end-$textPaint-$width-$alignment-$textDir-$spacingMul
t-$spacingAdd-$breakStrategy-$justificationMode"
```

Wrapping it up 📦

The final implementation provides us with an idiomatic (and, hopefully, performant) way of drawing multiline text to `Canvas`, which feels at home amongst other `Canvas` functions. It also includes full backwards compatibility and all of the available `StaticLayout` properties. Using Kotlin extension functions, named parameters and operator overloading has greatly reduced the amount of code and made the end result easier to use.

```
1  package com.nickrout.canvasmultilinetext
2
3  import android.graphics.Canvas
4  import android.os.Build
5  import android.support.annotation.RequiresApi
6  import android.text.*
7  import androidx.core.graphics.withTranslation
8  import androidx.core.util.lruCache
9
10 @RequiresApi(Build.VERSION_CODES.O)
11 fun Canvas.drawMultilineText(
12         text: CharSequence,
13         textPaint: TextPaint,
```

```
        y: Float,
        start: Int = 0,
        end: Int = text.length,
        alignment: Layout.Alignment = Layout.Alignment.ALIGN_NORMAL,
        textDir: TextDirectionHeuristic = TextDirectionHeuristics.FIRSTSTRONG_LTR,
        spacingMult: Float = 1f,
        spacingAdd: Float = 0f,
        includePad: Boolean = true,
        ellipsizedWidth: Int = width,
        ellipsize: TextUtils.TruncateAt? = null,
        maxLines: Int = Int.MAX_VALUE,
        breakStrategy: Int = Layout.BREAK_STRATEGY_SIMPLE,
        hyphenationFrequency: Int = Layout.HYPHENATION_FREQUENCY_NONE,
        justificationMode: Int = Layout.JUSTIFICATION_MODE_NONE) {

    val cacheKey = "$text-$start-$end-$textPaint-$width-$alignment-$textDir-" +
            "$spacingMult-$spacingAdd-$includePad-$ellipsizedWidth-$ellipsize-" +
            "$maxLines-$breakStrategy-$hyphenationFrequency-$justificationMode"

    val staticLayout = StaticLayoutCache[cacheKey] ?:
        StaticLayout.Builder.obtain(text, start, end, textPaint, width)
            .setAlignment(alignment)
            .setTextDirection(textDir)
            .setLineSpacing(spacingAdd, spacingMult)
            .setIncludePad(includePad)
            .setEllipsizedWidth(ellipsizedWidth)
            .setEllipsize(ellipsize)
            .setMaxLines(maxLines)
            .setBreakStrategy(breakStrategy)
            .setHyphenationFrequency(hyphenationFrequency)
            .setJustificationMode(justificationMode)
            .build().apply { StaticLayoutCache[cacheKey] = this }

    staticLayout.draw(this, x, y)
}

@RequiresApi(Build.VERSION_CODES.M)
fun Canvas.drawMultilineText(
        text: CharSequence,
        textPaint: TextPaint,
        width: Int,
        x: Float,
        y: Float,
        start: Int = 0,
        end: Int = text.length,
        alignment: Layout.Alignment = Layout.Alignment.ALIGN_NORMAL
```

```
        spacingAdd: Float = 0f,
        includePad: Boolean = true,
        ellipsizedWidth: Int = width,
        ellipsize: TextUtils.TruncateAt? = null,
        maxLines: Int = Int.MAX_VALUE,
        breakStrategy: Int = Layout.BREAK_STRATEGY_SIMPLE,
        hyphenationFrequency: Int = Layout.HYPHENATION_FREQUENCY_NONE) {

    val cacheKey = "$text-$start-$end-$textPaint-$width-$alignment-$textDir-" +
            "$spacingMult-$spacingAdd-$includePad-$ellipsizedWidth-$ellipsize-" +
            "$maxLines-$breakStrategy-$hyphenationFrequency"

    val staticLayout = StaticLayoutCache[cacheKey] ?:
        StaticLayout.Builder.obtain(text, start, end, textPaint, width)
            .setAlignment(alignment)
            .setTextDirection(textDir)
            .setLineSpacing(spacingAdd, spacingMult)
            .setIncludePad(includePad)
            .setEllipsizedWidth(ellipsizedWidth)
            .setEllipsize(ellipsize)
            .setMaxLines(maxLines)
            .setBreakStrategy(breakStrategy)
            .setHyphenationFrequency(hyphenationFrequency)
            .build().apply { StaticLayoutCache[cacheKey] = this }

    staticLayout.draw(this, x, y)
}

fun Canvas.drawMultilineText(
        text: CharSequence,
        textPaint: TextPaint,
        width: Int,
        x: Float,
        y: Float,
        start: Int = 0,
        end: Int = text.length,
        alignment: Layout.Alignment = Layout.Alignment.ALIGN_NORMAL,
        spacingMult: Float = 1f,
        spacingAdd: Float = 0f,
        includePad: Boolean = true,
        ellipsizedWidth: Int = width,
        ellipsize: TextUtils.TruncateAt? = null) {

    val cacheKey = "$text-$start-$end-$textPaint-$width-$alignment-" +
            "$spacingMult-$spacingAdd-$includePad-$ellipsizedWidth-$ellipsize"
```

```
    val staticLayout = StaticLayoutCache[cacheKey] ?:
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            StaticLayout.Builder.obtain(text, start, end, textPaint, width)
                    .setAlignment(alignment)
                    .setLineSpacing(spacingAdd, spacingMult)
                    .setIncludePad(includePad)
                    .setEllipsizedWidth(ellipsizedWidth)
                    .setEllipsize(ellipsize)
                    .build()
        } else {
            StaticLayout(text, start, end, textPaint, width, alignment,
                    spacingMult, spacingAdd, includePad, ellipsize, ellipsizedWidth)
                    .apply { StaticLayoutCache[cacheKey] = this }
        }

    staticLayout.draw(this, x, y)
}

private fun StaticLayout.draw(canvas: Canvas, x: Float, y: Float) {
    canvas.withTranslation(x, y) {
        draw(this)
    }
}

private object StaticLayoutCache {

    private const val MAX_SIZE = 50 // Arbitrary max number of cached items
    private val cache = lruCache<String, StaticLayout>(MAX_SIZE)

    operator fun set(key: String, staticLayout: StaticLayout) {
        cache.put(key, staticLayout)
    }

    operator fun get(key: String): StaticLayout? {
        return cache[key]
    }
```

• • •

then I'd love to hear from you!

Find me on Twitter @ricknout

Android Android Development Android App Development AndroidDev Android Apps