

Lesson 1.3: Text and scrolling views

Introduction

The [TextView](#) class is a subclass of the [View](#) class that displays text on the screen. You can control how the text appears with TextView attributes in the XML layout file. This practical shows how to work with multiple TextView elements, including one in which the user can scroll its contents vertically.

If you have more information than fits on the device's display, you can create a *scrolling view* so that the user can scroll vertically by swiping up or down, or horizontally by swiping right or left.

You would typically use a scrolling view for news stories, articles, or any lengthy text that doesn't completely fit on the display. You can also use a scrolling view to enable users to enter multiple lines of text, or to combine UI elements (such as a text field and a button) within a scrolling view.

The [ScrollView](#) class provides the layout for the scrolling view. `ScrollView` is a subclass of [FrameLayout](#). Place only *one* view as a child within it—a child view contains the entire contents to scroll. This child view may itself be a [ViewGroup](#) (such as [LinearLayout](#)) containing UI elements.

Complex layouts may suffer performance issues with child views such as images. A good choice for a `View` within a `ScrollView` is a `LinearLayout` that is arranged in a vertical orientation, presenting items that the user can scroll through (such as `TextView` elements).

With a `ScrollView`, all of the UI elements are in memory and in the view hierarchy even if they aren't displayed on screen. This makes `ScrollView` ideal for scrolling pages of free-form text smoothly, because the text is already in memory. However, `ScrollView` can use up a lot of memory, which can affect the performance of the rest of your app. To display long lists of items that users can add to, delete from, or edit, consider using a [RecyclerView](#), which is described in a separate lesson.

What you should already know

You should be able to:

- Create a Hello World app with Android Studio.
- Run an app on an emulator or a device.
- Implement a `TextView` in a layout for an app.
- Create and use string resources.

What you'll learn

- How to use XML code to add multiple `TextView` elements.
- How to use XML code to define a scrolling `View`.
- How to display free-form text with some HTML formatting tags.
- How to style the `TextView` background color and text color.
- How to include a web link in the text.

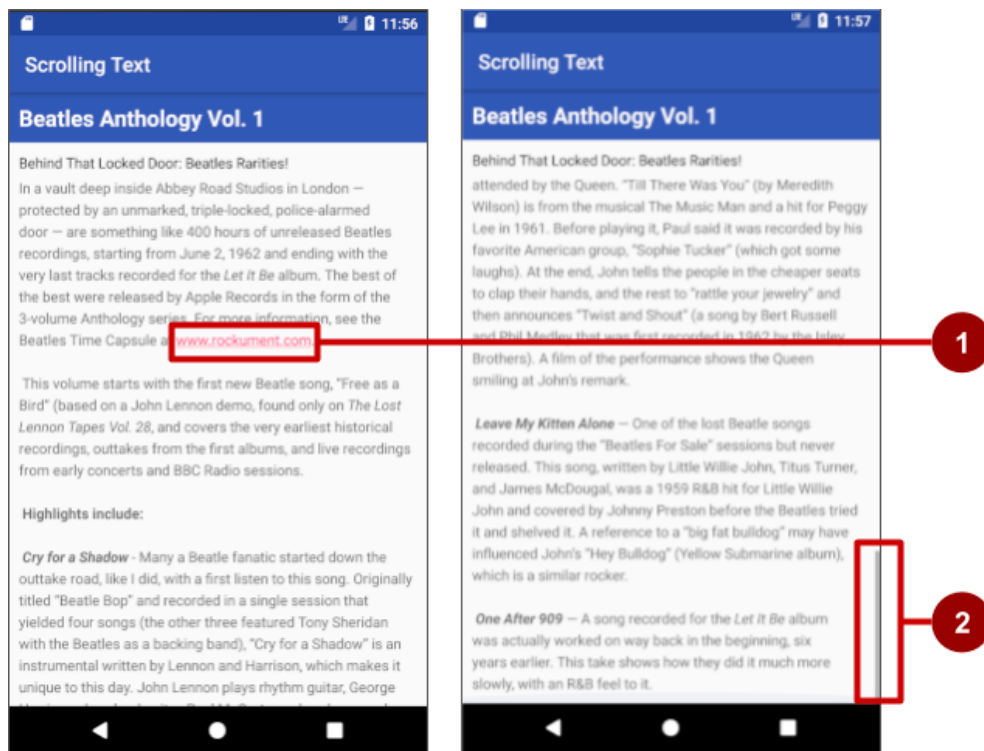
What you'll do

- Create the `ScrollingText` app.
- Change the `ConstraintLayout` `ViewGroup` to `RelativeLayout`.
- Add two `TextView` elements for the article heading and subheading.
- Use `TextAppearance` styles and colors for the article heading and subheading.

- Use HTML tags in the text string to control formatting.
- Use the `lineSpacingExtra` attribute to add line spacing for readability.
- Add a `ScrollView` to the layout to enable scrolling a `TextView` element.
- Add the `autoLink` attribute to enable URLs in the text to be active and clickable.

App overview

The Scrolling Text app demonstrates the [ScrollView](#) UI component. `ScrollView` is a `ViewGroup` that in this example contains a `TextView`. It shows a lengthy page of text—in this case, a music album review—that the user can scroll vertically to read by swiping up and down. A scroll bar appears in the right margin. The app shows how you can use text formatted with minimal HTML tags for setting text to bold or italic, and with new-line characters to separate paragraphs. You can also include active web links in the text.

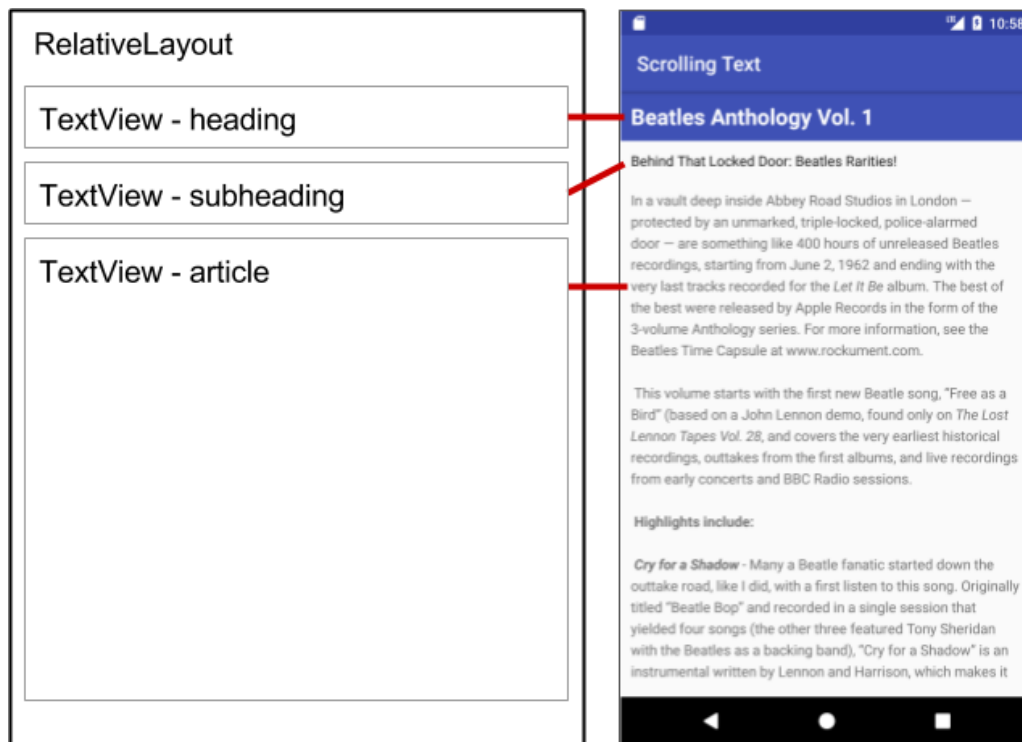


In the above figure, the following appear:

1. An active web link embedded in free-form text
2. The scroll bar that appears when scrolling the text

Task 1: Add and edit TextView elements

In this practical, you will create an Android project for the ScrollingText app, add TextView elements to the layout for an article title and subtitle, and change the existing “Hello World” TextView element to show a lengthy article. The figure below is a diagram of the layout.



You will make all these changes in the XML code and in the `strings.xml` file. You will edit the XML code for the layout in the Text pane, which you show by clicking the **Text** tab, rather than clicking the **Design** tab for the Design pane. Some changes to UI elements and attributes are easier to make directly in the Text pane using XML source code.

1.1 Create the project and TextView elements

In this task you will create the project and the TextView elements, and use TextView attributes for styling the text and background.

Tip: To learn more about these attributes, see the [TextView](#) reference.

1. In Android Studio create a new project with the following parameters:

Attribute	Value
Application Name	Scrolling Text
Company Name	android.example.com (or your own domain)
Phone and Tablet Minimum SDK	API15: Android 4.0.3 IceCreamSandwich
Template	Empty Activity
Generate Layout File checkbox	Selected
Backwards Compatibility (AppCompat) checkbox	Selected

2. In the **app > res > layout** folder in the **Project > Android** pane, open the **activity_main.xml** file, and click the **Text** tab to see the XML code.

At the top, or *root*, of the View hierarchy is the [ConstraintLayout](#) ViewGroup:

```
android.support.constraint.ConstraintLayout
```

3. Change this ViewGroup to [RelativeLayout](#). The second line of code now looks something like this:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

RelativeLayout lets you place UI elements relative to each other, or relative to the parent RelativeLayout itself.

The default “Hello World” TextView element created by the Empty Layout template still has constraint attributes (such as `app:layout_constraintBottom_toBottomOf="parent"`). Don't worry—you will remove them in a subsequent step.

4. Delete the following line of XML code, which is related to ConstraintLayout:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

The block of XML code at the top now looks like this:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.android.scrollingtext.MainActivity">
```

5. Add a TextView element above the “Hello World” TextView by entering **<TextView**. A TextView block appears that ends with `</>` and shows the `layout_width` and `layout_height` attributes, which are required for the TextView.
6. Enter the following attributes for the TextView. As you enter each attribute and value, suggestions appear to complete the attribute name or value.

TextView #1 attribute	Value
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:id	"@+id/article_heading"
android:background	"@color/colorPrimary"
android:textColor	"@android:color/white"
android:padding	"10dp"
android:textAppearance	"@android:style/TextAppearance.DeviceDefault.Large"
android:textStyle	"bold"
android:text	"Article Title"

7. Extract the string resource for the android:text attribute's hardcoded string "Article Title" in the TextView to create an entry for it in **strings.xml**.

Place the cursor on the hardcoded string, press Alt-Enter (Option-Enter on the Mac), and select **Extract string resource**. Make sure that the **Create the resource in directories** option is selected, and then edit the resource name for the string value to **article_title**.

String resources are described in detail in the [String Resources](#).

8. Extract the dimension resource for the android:padding attribute's hardcoded string "10dp" in the TextView to create **dimens.xml** and add an entry to it.

Place the cursor on the hardcoded string, press Alt-Enter (Option-Enter on the Mac), and select **Extract dimension resource**. Make sure that the **Create the resource in directories** option is selected, and then edit the Resource name to **padding_regular**.

9. Add another TextView element above the "Hello World" TextView and below the TextView you created in the previous steps. Add the following attributes to the TextView:

TextView #2 Attribute	Value
layout_width	"match_parent"

layout_height	"wrap_content"
android:id	"@+id/article_subheading"
android:layout_below	"@id/article_heading"
android:padding	"@dimen/padding_regular"
android:textAppearance	"@android:style/TextAppearance.DeviceDefault"
android:text	"Article Subtitle"

Because you extracted the dimension resource for the "10dp" string to `padding_regular` in the previously created `TextView`, you can use `"@dimen/padding_regular"` for the `android:padding` attribute in this `TextView`.

10. Extract the string resource for the `android:text` attribute's hardcoded string "Article Subtitle" in the `TextView` to **article_subtitle**.
11. In the "Hello World" `TextView` element, delete the `layout_constraint` attributes:

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

12. Add the following `TextView` attributes to the "Hello World" `TextView` element, and change the `android:text` attribute:

TextView Attribute	Value
android:id	"@+id/article"
android:layout_below	"@id/article_subheading"
android:lineSpacingExtra	"5sp"
android:padding	"@dimen/padding_regular"
android:text	Change to "Article text"

13. Extract the string resource for "Article text" to **article_text**, and extract the dimension resource for "5sp" to **line_spacing**.
14. Reformat and align the code by choosing **Code > Reformat Code**. It is a good practice to reformat and align your code so that it is easier for you and others to understand.

1.2 Add the text of the article

In a real app that accesses magazine or newspaper articles, the articles that appear would probably come from an online source through a content provider, or might be saved in advance in a database on the device.

For this practical, you will create the article as a single long string in the strings.xml resource.

1. In the **app > res > values** folder, open **strings.xml**.
2. Open any text file with a large amount of text, or open the [strings.xml file of the finished ScrollingText app](#).
3. Enter the values for the strings `article_title` and `article_subtitle` with either a made-up title and subtitle, or use the values in the strings.xml file of the finished ScrollingText app. Make the string values single-line text without HTML tags or multiple lines.
4. Enter or copy and paste text for the `article_text` string.

You can use the text in your text file, or use the text provided for the `article_text` string in the strings.xml file of the finished ScrollingText app. The only requirement for this task is that the text must be long enough so that it doesn't fit on the screen.

Keep in mind the following (refer to the figure below for an example):

- As you enter or paste text in the strings.xml file, the text lines don't wrap around to the next line—they extend beyond the right margin. This is the correct behavior—each new line of text starting at the left margin represents an entire paragraph. If you want the text in strings.xml to be wrapped, you can press Return to enter hard line endings, or format the text first in a text editor with hard line endings.
- Enter `\n` to represent the end of a line, and another `\n` to represent a blank line. You need to add end-of-line characters to keep paragraphs from running into each other.

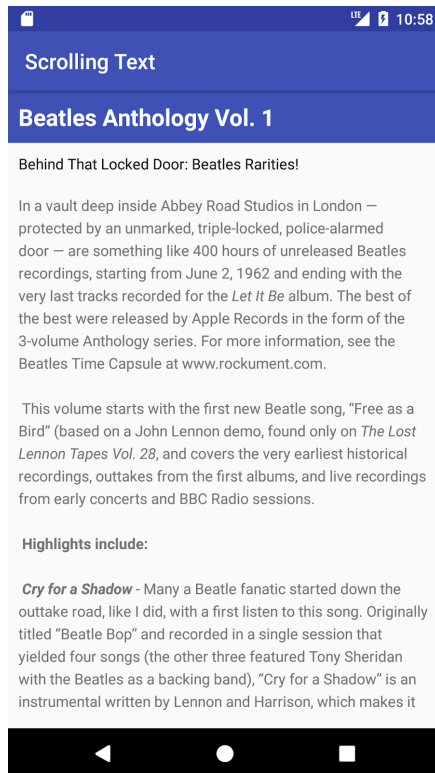
- If you have an apostrophe (') in your text, you must escape it by preceding it with a backslash (\'). If you have a double-quote in your text, you must also escape it (\"). You must also escape any other non-ASCII characters. See the [Formatting and styling](#) section of [String resources](#) for more details.
- Enter the HTML **** and **** tags around words that should be in bold.
- Enter the HTML **<i>** and **</i>** tags around words that should be in italics. If you use curled apostrophes within an italic phrase, replace them with straight apostrophes.
- You can combine bold and italics by combining the tags, as in **<i>... words...</i>**. Other HTML tags are ignored.
- Enclose The entire text within `<string name="article_text"> </string>` in the strings.xml file.
- Include a web link to test, such as **www.google.com**. (The example below uses **www.rockument.com**.) *Don't* use an HTML tag, because any HTML tags except the bold and italic tags are ignored and presented as text, which is not what you want.

```

1 <resources>
2   <string name="app_name">Scrolling Text</string>
3   <string name="article_title">Beatles Anthology Vol. 1</string>
4   <string name="article_subtitle">Behind That Locked Door: Beatles Rarities!</string>
5   <string name="article_text">In a vault deep inside Abbey Road Studios in London – protected by an unmarked, triple-lock
6     For more information, see the Beatles Time Capsule at www.rockument.com.
7   \n\n
8   This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on <i>The Lost
9   \n\n
10  <b>Highlights include:</b>
11  \n\n
12  <b><i>Cry for a Shadow</i></b> – Many a Beatle fanatic started down the outtake road, like I did, with a first listen to th
13  \n\n
14  <b><i>My Bonnie</i></b> and <b><i>Ain't She Sweet</i></b> – At the same session, the Beatles played on "My Bonnie" (the fi
15  \n\n
16  <b><i>Searchin</i></b> – A Jerry Leiber – Mike Stoller comedy song that was a hit for the Coasters in 1957, and a popular
17  \n\n
18  <b><i>Love Me Do</i></b> – An early version of the song, played a bit slower and with more of a blues feeling, and a cool
19  \n\n
20  <b><i>She Loves You – Till There Was You – Twist and Shout</i></b> – Live at the Princess Wales Theatre by Leicester Square
21  \n\n
22  <b><i>Leave My Kitten Alone</i></b> – One of the lost Beatle songs recorded during the "Beatles For Sale" sessions but neve
23  \n\n
24  <b><i>One After 909</i></b> – A song recorded for the <i>Let It Be</i> album was actually worked on way back in the beginn
25  </string>
26 </resources>
27
  
```

1.3 Run the app

Run the app. The article appears, but the user can't scroll the article because you haven't yet included a `ScrollView` (which you will do in the next task). Note also that tapping a web link does not currently do anything. You will also fix that in the next task.



Task 1 solution code

The `activity_main.xml` layout file looks like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
tools:context="com.example.android.scrollingtext.MainActivity">

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/article_heading"
    android:background="@color/colorPrimary"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_title"
    android:textAppearance=
        "@android:style/TextAppearance.DeviceDefault.Large"
    android:textColor="@android:color/white"
    android:textStyle="bold" />

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/article_subheading"
    android:layout_below="@id/article_heading"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_subtitle"
    android:textAppearance=
        "@android:style/TextAppearance.DeviceDefault" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/article"
    android:layout_below="@id/article_subheading"
    android:lineSpacingExtra="@dimen/line_spacing"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_text" />

</RelativeLayout>
```

Task 2: Add a ScrollView and an active web link

In the previous task you created the ScrollingText app with TextView elements for an article title, subtitle, and lengthy article text. You also included a web link, but the link is not yet active. You will add the code to make it active.

Also, the TextView by itself can't enable users to scroll the article text to see all of it. You will add a new ViewGroup called ScrollView to the XML layout that will make the TextView scrollable.

2.1 Add the autoLink attribute for active web links

Add the `android:autoLink="web"` attribute to the article TextView. The XML code for this TextView now looks like this:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/article"
    android:autoLink="web"
    android:layout_below="@id/article_subheading"
    android:lineSpacingExtra="@dimen/line_spacing"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_text" />
```

2.2 Add a ScrollView to the layout

To make a View (such as a TextView) scrollable, embed the View *inside* a ScrollView.

1. Add a **ScrollView** between the `article_subheading` `TextView` and the `article` `TextView`. As you enter **<ScrollView**, Android Studio automatically adds `</ScrollView>` at the end, and presents the `android:layout_width` and `android:layout_height` attributes with suggestions.
2. Choose **wrap_content** from the suggestions for both attributes.

The code for the two `TextView` elements and the `ScrollView` now looks like this:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/article_subheading"
    android:layout_below="@id/article_heading"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_subtitle"
    android:textAppearance=
        "@android:style/TextAppearance.DeviceDefault"/>

<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ScrollView>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/article"
    android:autoLink="web"
    android:layout_below="@id/article_subheading"
    android:lineSpacingExtra="@dimen/line_spacing"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_text" />
```

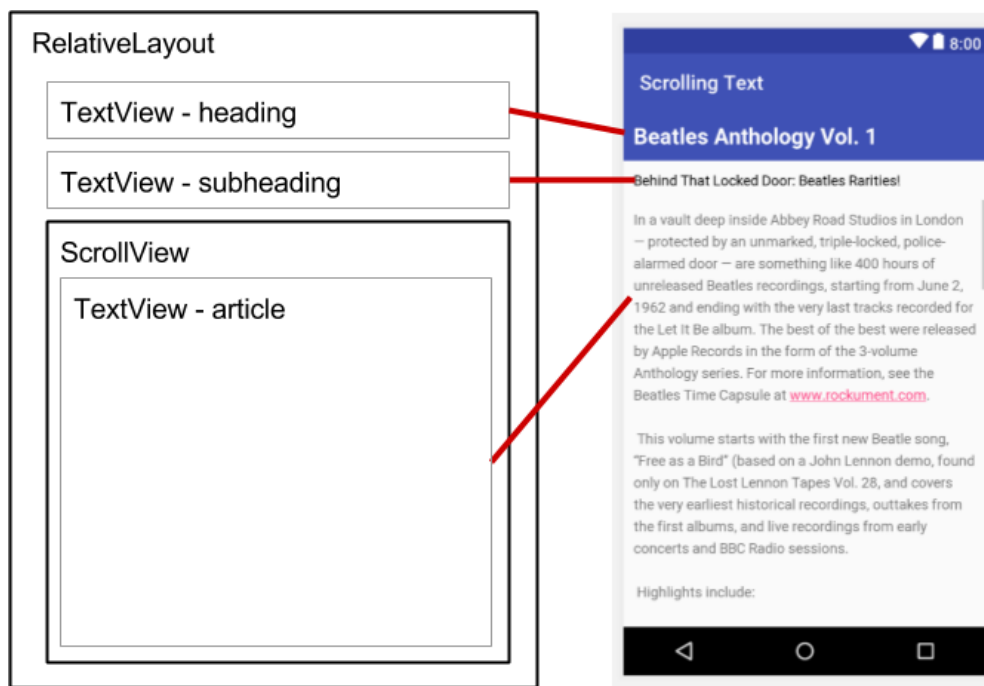
3. Move the ending `</ScrollView>` code *after* the `article` `TextView` so that the `article` `TextView` attributes are entirely inside the `ScrollView`.
4. Remove the following attribute from the `article` `TextView` and add it to the `ScrollView`:

```
android:layout_below="@id/article_subheading"
```

With the above attribute, the `ScrollView` element will appear below the article subheading. The article is inside the `ScrollView` element.

5. Choose **Code > Reformat Code** to reformat the XML code so that the article `TextView` now appears indented inside the `<ScrollView>` code.
6. Click the **Preview** tab on the right side of the layout editor to see a preview of the layout.

The layout now looks like the right side of the following figure:



2.3 Run the app

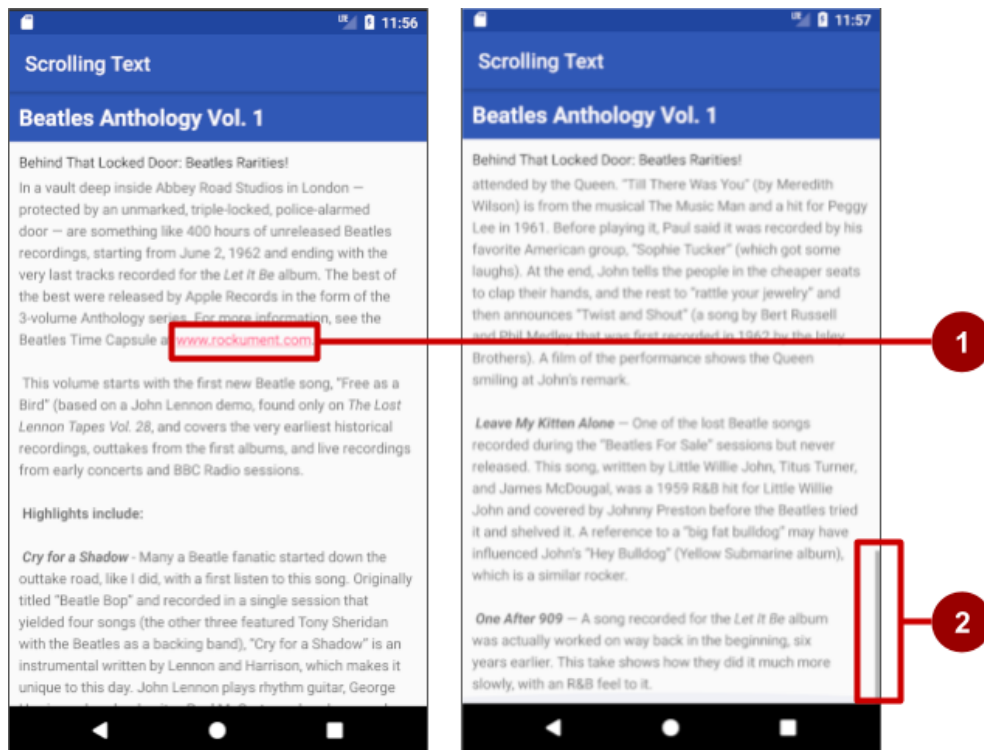
To examine how the text scrolls:

1. Run the app on a device or emulator.

Swipe up and down to scroll the article. The scroll bar appears in the right margin as you scroll.

Tap the web link to go to the web page. The `android:autoLink` attribute turns any recognizable URL in the TextView (such as `www.rockument.com`) into a web link.

2. Rotate your device or emulator while running the app. Notice how the scrolling view widens to use the full display and still scrolls properly.
3. Run the app on a tablet or tablet emulator. Notice how the scrolling view widens to use the full display and still scrolls properly.



In the above figure, the following appear:

1. An active web link embedded in free-form text
2. The scroll bar that appears when scrolling the text

Task 2 solution code

The XML code for the layout with the scroll view is as follows:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.android.scrollingtext.MainActivity">

    <TextView
        android:id="@+id/article_heading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_title"
        android:textAppearance=
            "@android:style/TextAppearance.DeviceDefault.Large"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/article_subheading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_heading"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_subtitle"
        android:textAppearance=
            "@android:style/TextAppearance.DeviceDefault" />

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_subheading">

        <TextView
            android:id="@+id/article"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:autoLink="web"
```

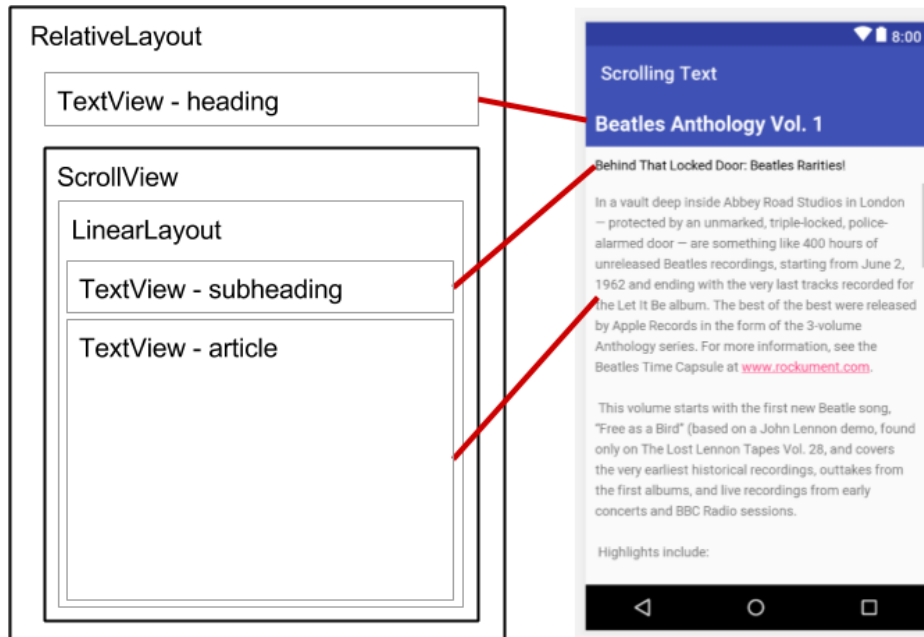
```
        android:lineSpacingExtra="@dimen/line_spacing"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_text" />

    </ScrollView>
</RelativeLayout>
```

Task 3: Scroll multiple elements

As noted before, a `ScrollView` can contain only one child View (such as the article `TextView` you created). However, that View can be another `ViewGroup` that contains View elements, such as [LinearLayout](#). You can *nest* a `ViewGroup` such as `LinearLayout` *within* the `ScrollView`, thereby scrolling everything that is inside the `LinearLayout`.

For example, if you want the subheading of the article to scroll along with the article, add a `LinearLayout` within the `ScrollView`, and move the subheading and article into the `LinearLayout`. The `LinearLayout` becomes the single child View in the `ScrollView` as shown in the figure below, and the user can scroll the entire `LinearLayout`: the subheading and the article.



3.1 Add a LinearLayout to the ScrollView

1. Open the **activity_main.xml** file of the ScrollingText app project, and select the **Text** tab to edit the XML code (if it is not already selected).
2. Add a **LinearLayout** above the article **TextView** within the **ScrollView**. As you enter **<LinearLayout>**, Android Studio automatically adds **</LinearLayout>** to the end, and presents the **android:layout_width** and **android:layout_height** attributes with suggestions. Choose **match_parent** and **wrap_content** from the suggestions for its width and height, respectively. The code at the beginning of the **ScrollView** now looks like this:

```
<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/article_subheading">

    <LinearLayout
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"></LinearLayout>

    <TextView
        android:id="@+id/article"
```

You use `match_parent` to match the width of the parent `ViewGroup`. You use `wrap_content` to resize the `LinearLayout` so it is just big enough to enclose its contents.

3. Move the ending `</LinearLayout>` code *after* the article `TextView` but *before* the closing `</ScrollView>`.

The `LinearLayout` now includes the article `TextView`, and is completely inside the `ScrollView`.

4. Add the `android:orientation="vertical"` attribute to the `LinearLayout` to set its orientation to vertical.
5. Choose **Code > Reformat Code** to indent the code correctly.

The `LinearLayout` now looks like this:

```
<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/article_subheading">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/article"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:autoLink="web"
            android:lineSpacingExtra="@dimen/line_spacing"
            android:padding="@dimen/padding_regular"
```

```
        android:text="@string/article_text" />
    </LinearLayout>

</ScrollView>
```

3.2 Move UI elements within the LinearLayout

The LinearLayout now has only one UI element—the article TextView. You want to include the article_subheading TextView in the LinearLayout so that both will scroll.

1. To move the article_subheading TextView, select the code, choose **Edit > Cut**, click above the article TextView inside the LinearLayout, and choose **Edit > Paste**.
2. Remove the android:layout_below="@id/article_heading" attribute from the article_subheading TextView. Because this TextView is now within the LinearLayout, this attribute would conflict with the LinearLayout attributes.
3. Change the ScrollView layout attribute from android:layout_below="@id/article_subheading" to android:layout_below="@id/article_heading". Now that the subheading is part of the LinearLayout, the ScrollView must be placed below the heading, not the subheading.

The XML code for the ScrollView is now as follows:

```
<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/article_heading">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/article_subheading"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
        android:padding="@dimen/padding_regular"
        android:text="@string/article_subtitle"
        android:textAppearance=
            "@android:style/TextAppearance.DeviceDefault" />

        <TextView
            android:id="@+id/article"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:autoLink="web"
            android:lineSpacingExtra="@dimen/line_spacing"
            android:padding="@dimen/padding_regular"
            android:text="@string/article_text" />
    </LinearLayout>
</ScrollView>
```

4. Run the app.

Swipe up and down to scroll the article, and notice that the subheading now scrolls along with the article while the heading stays in place.