|  |
| --- |
| Android Tutorial – Part 2 |

|  |
| --- |
| 6-2-2018 |



Table of Contents

[Introduction 2](#_Toc515829150)

[Android Studio’s UI Design View - Layout Editor 2](#_Toc515829151)

[The Design View of Layout Editor 3](#_Toc515829152)

[Toolbar items to Change the preview appearance 4](#_Toc515829153)

[Edit View attributes 5](#_Toc515829154)

[The Text View of Layout Editor 6](#_Toc515829155)

[Android Views and View Groups 7](#_Toc515829156)

[Layout files 9](#_Toc515829157)

[Create a Layout file 9](#_Toc515829158)

[Rename a file in Android Studio 11](#_Toc515829159)

[Compare two files using Android Studio 13](#_Toc515829160)

[Delete a file in Android Studio 15](#_Toc515829161)

[Design a User interface 16](#_Toc515829162)

[Different Types of Layouts 17](#_Toc515829163)

[Attributes of UI components 17](#_Toc515829164)

[Usage of string.xml file 17](#_Toc515829165)

[Usage of colours.xml file 17](#_Toc515829166)

[Usage of styles.xml file 17](#_Toc515829167)

[Access and Modify UI from Activity class 17](#_Toc515829168)

[Toasts 17](#_Toc515829169)

[Intents 17](#_Toc515829170)

[References 17](#_Toc515829171)

# Introduction

This is the part two of the android tutorial series. It is a continuation from last week. In order to follow this successfully, it is required to have,

* A basic understanding given about android in last session.
* The environment set up.
* The project created during last, opened in Android Studio.
* AVD or an Actual device ready for app deployment.

To catch up, in the last session,

* An introduction to android was given
* Android studio was introduced
* It was shown how to create a new android project using android studio.
* The structure of an android project was explained.
* A brief introduction about each component of the project was given.
* Shown how to run an android app in an Android virtual device (AVD) and a Real device.

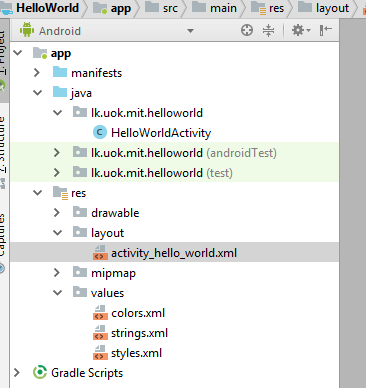
With that knowledge in hand, in this session below areas will be covered,

* Android Studio’s UI Design View (Layout Editor)
* Android Views and View Groups
* Different Types of layouts
* Design a User interface
  + Attributes of UI components
  + Usage of string.xml file
  + Usage of colours.xml file
  + Usage of styles.xml file
* Access and Modify UI from Activity class
* Toasts
* Intents

# Android Studio’s UI Design View - Layout Editor

Since the main focus of this tutorial is to show how to properly design user interfaces of android applications, as the first step, it’s required to get familiar with the Layout Editor of Android Studio first.

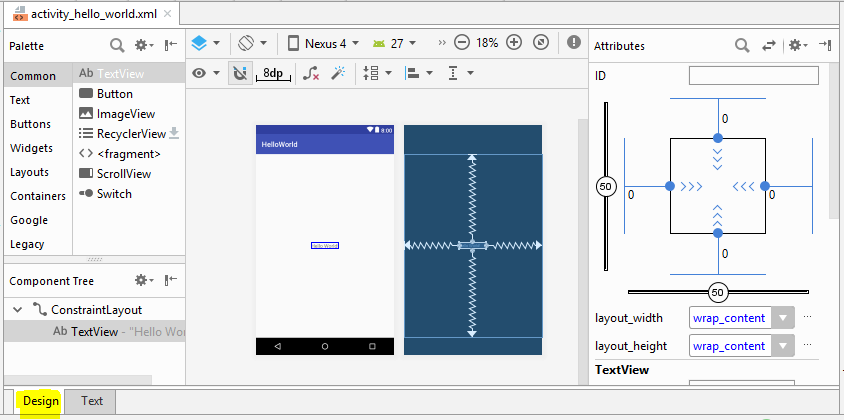
To open the layout editor, a layout xml file should be opened. If you could recall, we added an “Empty Activity” during creation of a new project, and named it as “HelloWorldActivity”. Along with that, an xml file was created as “**activity\_hello\_world.xml**”. This layout xml file could be seen at res->layout folder.



* Click on “activity\_hello\_world.xml” to open it in the Layout Editor of Android Studio.
* The layout editor view appears as below, there are two main views as;
  + **Design** – Facilitate UI design by providing Drag and Drop from palette
  + **Text** – Facilitate UI design by allowing to edit XML

## The Design View of Layout Editor

* The **Design View** of Layout Editor and its crucial items are marked and listed below,



**View Structure Here**

**Edit Attributes Here**

**Drop Here**

**Drag From Here**

* **Palette**: List of **views** and **view groups** that you can **drag** into your **layout**, in a categorized view
* **Component Tree**: View hierarchy for your layout.
  + Note there are only two components in the layout which was auto generated for our “HelloWorld” activity
    - ConstraintLayout – the parent layout or “View Group”
    - TextView – a text view with text “HelloWorld”, which is indicated as a child view under parent ConstraintLayout in Component Tree
* **Toolbar**: Buttons to configure your layout appearance in the editor and to change some layout attributes.
* **Design editor**: Layout in Design or Blueprint view, or both, allows to view either real world design of UI, or the blueprint, or both as shown above.
* **Attributes**: Controls for the selected view's attributes, this lets edit attributes of the selected view of the layout file

### Toolbar items to Change the preview appearance

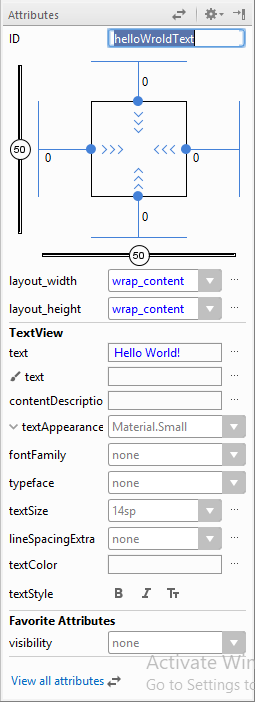
The buttons in the top row of the design editor allow you to configure the appearance of your layout in the editor. This toolbar is also available in the text editor's Preview window.



Corresponding to the numbers in above figure, the buttons available are as follows:

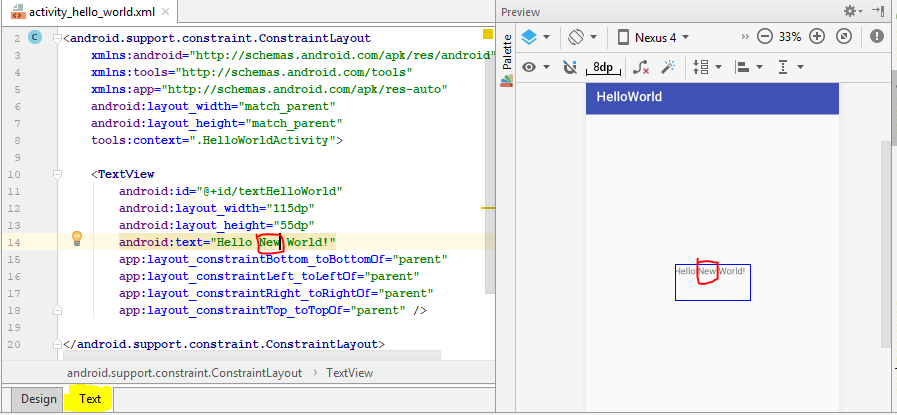
* **Design and blueprint**: Select how you'd like to view your layout in the editor; select either the
  + **Design view** (a real-world preview of the layout),
  + **The Blueprint view** (only outlines for each view),
  + OR Design + Blueprint for both side by side.
* **Screen orientation and layout variants**: Select between
  + - Landscape orientation
    - portrait screen orientation,
  + Also allows to choose other screen modes for which your app provides alternative layouts, such as night mode.
  + This menu also contains commands for creating a new layout variant.
* **Device type and size**: Select the
  + - device type (phone/tablet, Android TV, or Wear OS) and
    - Screen configuration (size and density).
  + Can select from several pre-configured device types and own AVD definitions
  + Also allows to start a new AVD by selecting Add Device Definition from the list.
* **API version**: Select the version of Android on which to preview your layout.
* **App theme**: Select which UI theme to apply to the preview. (This works only for supported layout styles; thus many themes in this list result in an error.)
* **Language**: Select the language to show for your UI strings.
  + This list displays only the languages available in your string resources.
  + If you'd like to edit your translations, click Edit Translations from the drop-down menu.
* **Zoom Out** – allows to zoom out the design view
* **Zoom level** – displays the current zoom level
* **Zoom In** – allows to zoom in the design view
* **Zoom to Fit** – Automatically select the appropriate zoom level based on window size to fit the screen
* **Issues Indication** – notifies (turns red) if there are any issues in the current UI design.

### Edit View attributes

* Each UI view of a layout has a set of attributes.
  + E.g.:- id, with, height, color, text, etc.
* These can be edited either in “Attributes” window or in “Text” view by editing xml
* The “Attribute” Window is available only when the design editor is open, so be sure you've selected the Design tab at the bottom of the window.
* When a view is selected, either by,
  + Clicking it in the Component Tree or
  + Clicking it in the design editor
* The Attributes window will look like below when the TextView “Hello Wrold” is seleted,  
    
  
* **ID** – the id attribute is used to **uniquely** **identify** a **view**.
  + ID attribute is optional.
  + An **android:id** attribute is required to be added to any View component in the layout file only if you want a **reference** to it
    - Access the view in Java code.
    - Access the view when defining layout in XML.
* **View inspector** - with controls for width/height style, margins, and bias (**available only for views in a ConstraintLayout**) in a Responsive UI.
* **Common attributes** - A list of common attributes for the selected view.
  + To see all available attributes, click View all attributes at the top of the window.
* **Favorite attributes** - Favorite attributes you've selected, most commonly used attributes
  + To add attributes to this, click View all attribute Favorite attributes and then click the star that appears when you hover your mouse over the left side of an attribute name.
* **View all attributes** - To search for a specific view attribute, click View all attributes and then click Search at the top of the window.

## The Text View of Layout Editor

* The **Text View** of Layout Editor and its crucial items are marked and listed below,



* This view mainly consists of two parts as shown above,
  + XML editor – allows to edit the xml code of layout
  + Preview – same as in the design view, show the real design of UI
* Note the change in text of “Hello World” text box, the **android:text** attribute has been **modified**  using text view
* Some may find its comfortable to design UI by editing the xml file in text view and some may find its comfortable to use the design view, choose according to your preference and always can be used interchangeably

# Android Views and View Groups

Activities make up the bulk of the graphical user interface associated with Android applications. In Android, there are two ways to construct Activities.

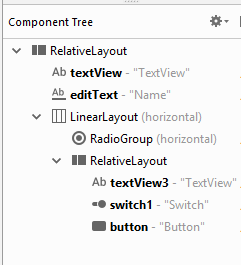
1. Programmatically add View and ViewGroup components (i.e. “widgets”) to the Activity in Java program code.
   * The programmatic approach is typically reserved for more dynamic and/or complicated UI scenarios.
2. Use XML layout files.
   * The typical and most common way to construct an Activity.

Composition of an android UI is a combination of **View** s and **ViewGroup** s

* The user interface for an Android app is built using a hierarchy of
  + **Layouts** (**ViewGroup**) objects
    - invisible containers that control how its child views are positioned on the screen
    - A layout manager is responsible for the layout of itself and its child views.
    - The base class for these layout managers is the **android.view.ViewGroup** class.
    - Layout managers can be nested to create complex layouts.
    - The most relevant layout managers in Android are:
      * ConstraintLayout - provided by an extra library
      * LinearLayout
      * FrameLayout
      * RelativeLayout
      * GridLayout
    - All layout manager can be configured via attributes.
    - Children can also define attributes which may be evaluated by their parent layout.
  + **Widgets** (**View**) objects.
    - UI components such as buttons and text boxes
    - A view in Android represents a widget, e.g., a button, or a layout manager.
    - All views in Android extend the **android.view.View** class.
    - The Android SDK provides standard views,
      * E.g.:- the Button, TextView and EditText classes.
    - The main packages for views are:
      * **android.view** for all base classes
      * **android.widget** for the default widgets
    - Additional libraries provide more complex widgets, for example, **RecyclerView**

An Illustration of how ViewGroup objects form branches in the layout and contain View objects is given below,

In the “Component Tree” view of the Android Studio, above arrangement will look like below,



## Layout files

As mentioned above, there are two ways to define user interfaces in Android that is via Java code and via XML files in res/layout folder. Between these two, **defining layouts via XML** layout files is the **preferred** way. That is because;

1. It separates the programming logic from the layout definition.
2. It allows the definition of different layouts for different devices.

A layout resource file is referred to as layout. A layout specifies

* the ViewGroups,
* the Views,
* the relationships between Views and ViewGroups,
* the attributes of Views and ViewGroups

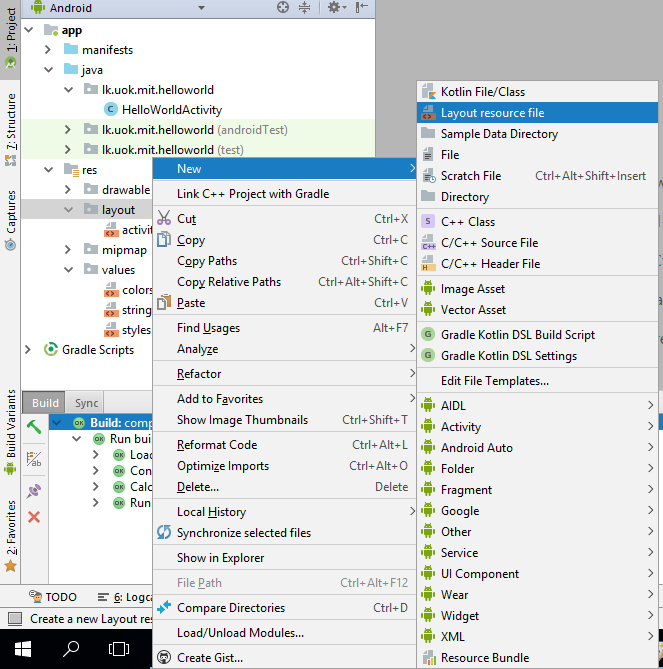
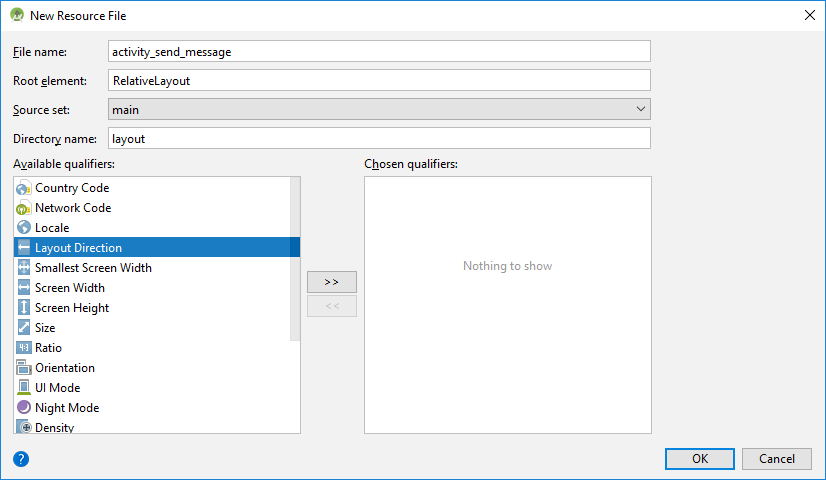
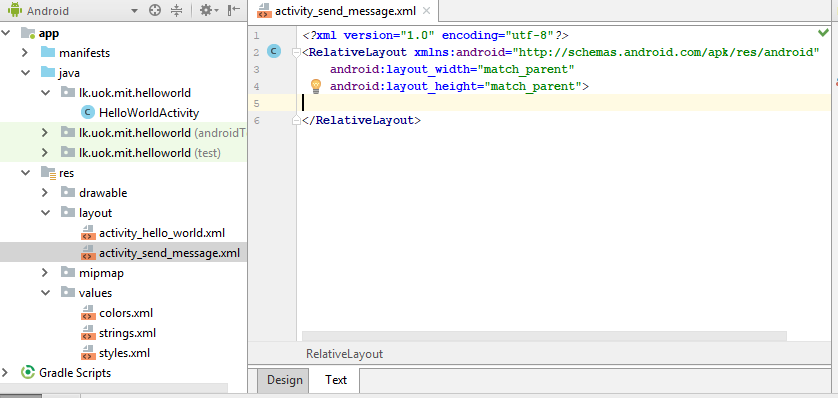
via an XML representation.

Previously a layout file (named “activity\_hello\_world.xml”) was auto generated for us, when we created an empty activity named “HelloWorldActivity”.   
To understand more about an activity and layout files, lets manually create a layout file, an activity class and bind them together as shown in the steps below,

### Create a Layout file

There are many ways to create a new layout file, below shown are 2 of them which are most commonly used, **practice both methods**, but **create only one xml** **file for the purpose of this tutorial!**

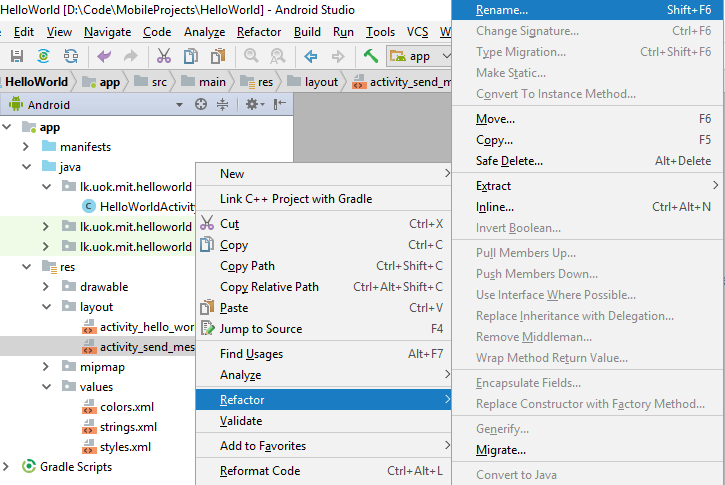
#### 1st way: - Using New🡪 “Layout resource file” option

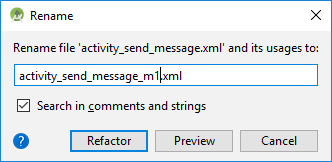
1. Step 1:- Got to **res/layout folder**, **right click** on it, select “**New**”🡪 “**Layout resource File**” as shown below,  
   
2. Step 2:- After above step, below screen will appear,  
   
3. Step 3:- Fill in the details in the dialog:
   1. **File name**: Type the name for the XML file (does not require the .xml suffix).
      1. Make it **“activity\_send\_message”**
   2. **Root element**: Select the root XML element for the layout file.
      1. Make it **“RelativeLayout”**
   3. **Source set**: Select the source set where you want to save the file.
      1. Leave it as **“main”**
   4. **Directory name**: Since we opened this dialog by right clicking on res/layout folder, “**layout**” should be auto filled.
      1. Leave it as “**layout**”.
   5. **Available qualifiers**: Instead of typing configuration qualifiers for your layout directory, you can add them by clicking a qualifier from the list on the left, and then clicking Add >>
      1. **No need to add additional qualifiers here**
4. Step 4:- Click “OK” in the dialog
5. Step 5:- Open the generated “activity\_send\_message.xml” inside res/layout folder and view its content  
   

### Rename a file in Android Studio

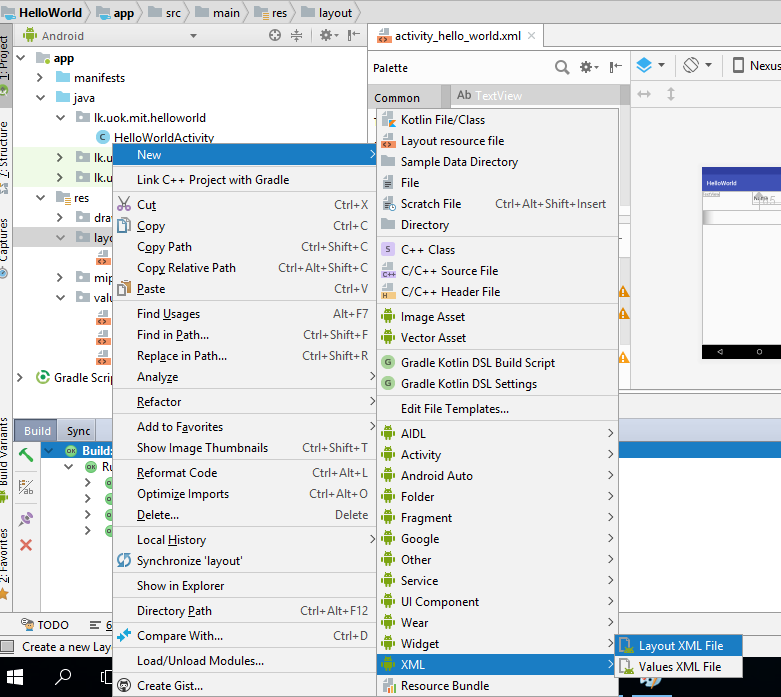
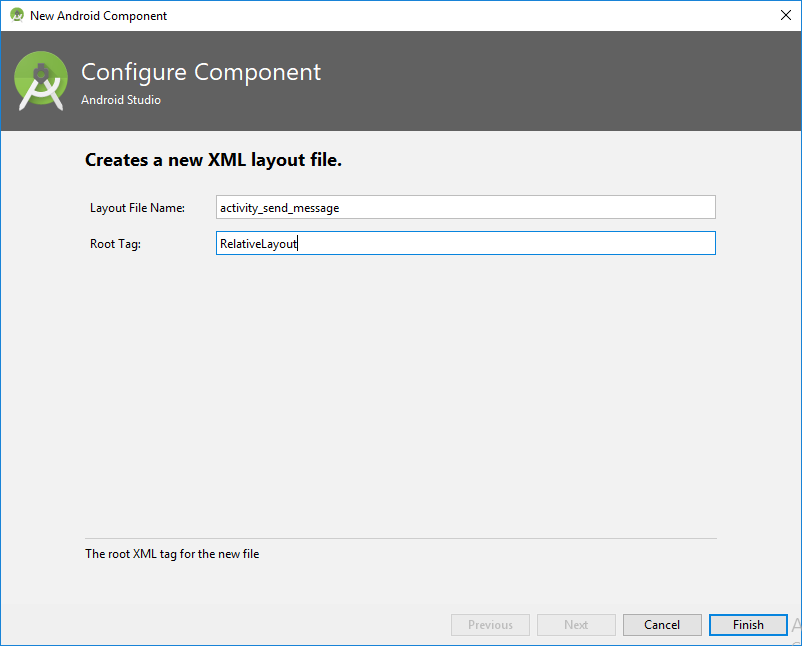
Rename the “activity\_send\_message.xml” file generated from above to “activity\_send\_message\_m1.xml” like shown below, this will be deleted later as we need only one xml file;

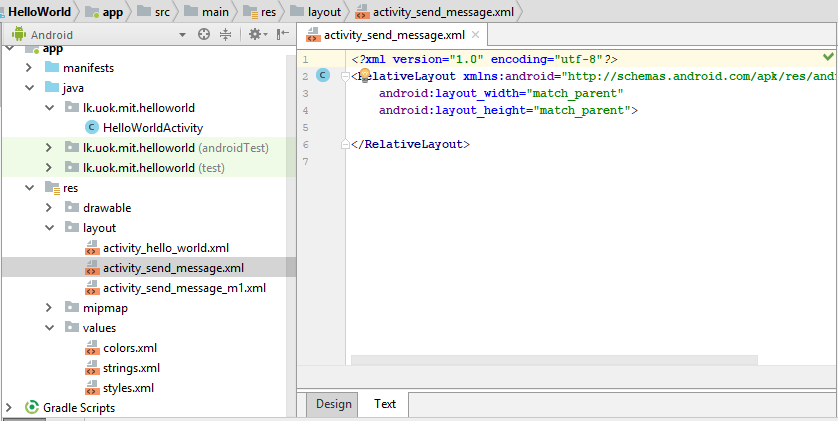
1. Step 1:- Go to **res/layout folder**, **right click** on “**activity\_send\_message.xml**”, select **Refactor**🡪”**Rename**” as shown, (or Shift + F6)



1. Step 2:- In the text box modify the file name to “**activity\_send\_message\_m1.xml**” and click “**Refactor**”  
   
   1. **Note**:- These are the steps to **Rename** any file inside Android Studio

#### 2nd way: - Using New🡪 XML🡪 “Layout XML file” option

1. Step 1:- Got to **res/layout folder**, **right click** on it, select **New**🡪**XML**🡪”**Layout XML File**” as shown,  
   
2. Step 2:- After above step, below screen will appear;  
   
3. Step 3:- Fill in the details in the dialog:
   1. **Layout File name**: Type the name for the XML file (does not require the .xml suffix).
      1. Make it **“activity\_send\_message”**
   2. **Root Tag**: Select the root XML element for the layout file.
      1. Make it **“RelativeLayout”**
4. Step 4:- Click “Finish” in the dialog
5. Step 5:- Open the generated “activity\_send\_message.xml” inside res/layout folder and view its content



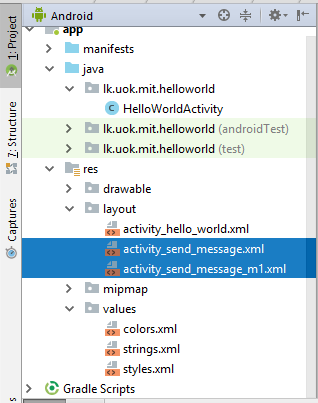
### Compare two files using Android Studio

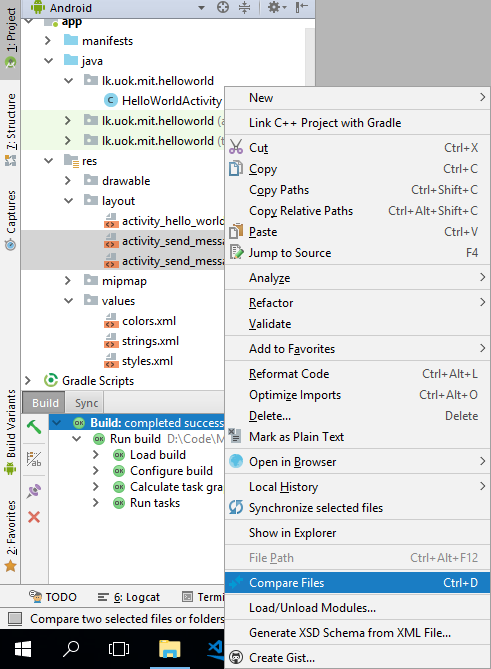
Above it is shown how to create a layout xml file with “RelativeLayout” as its root element, using two different methods.

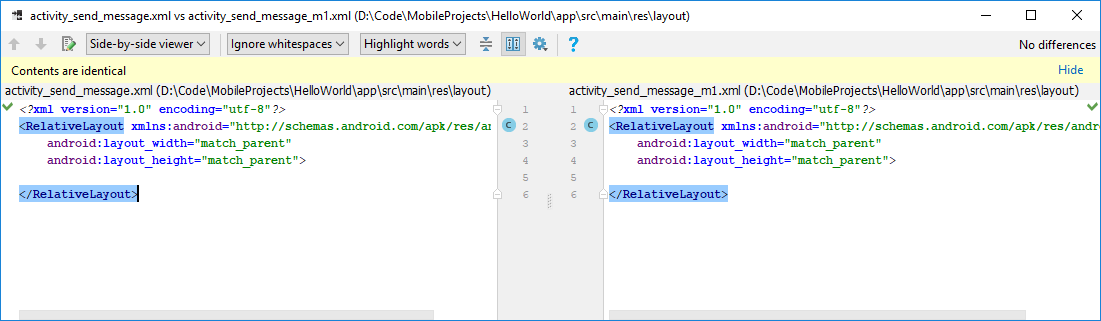
Now you should have below two files inside your **res/layout** folder as shown in above figure

1. activity\_send\_message\_m1.xml – file created using first approach and which was renamed later
2. activity\_send\_message.xml - file created using second approach

Now let’s compare these two files to check if there are any differences between them. To compare two files in Android Studio, follow the steps given below,

1. Step 1:- Select both files in “Project Explorer” windows by clicking on them while pressing “Shift Key” in the keyboard, as shown below;  
   
2. Step 2:- Right click on selected files “**blue area**”, and select “**Compare Files**” from menu, or press (Ctrl + D) in keyboard while selecting both files;



1. Step 3:- After the step 2; below window will appear, which will display two files side by side, with a comparison  
   

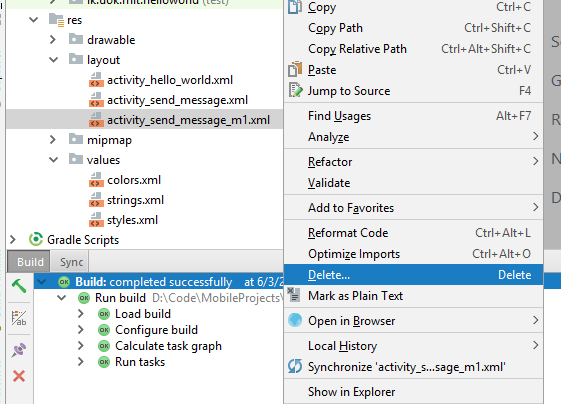
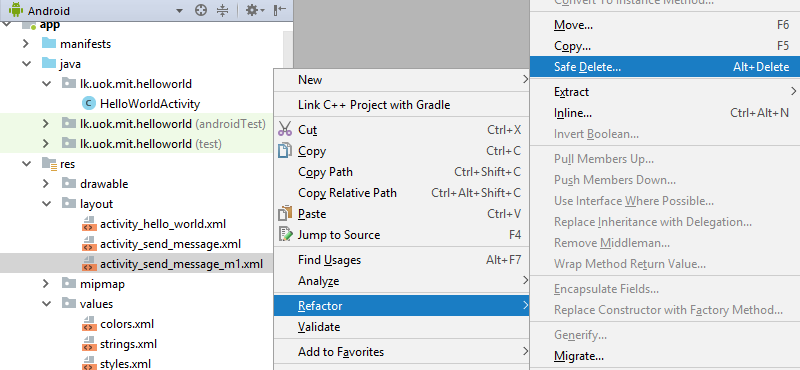
* As it is shown in above image, there are no differences between the two files.
* Therefore we can conclude that from both approaches mentioned above, a similar layout file could be generated.
* To add a further note here, first approach can be used to generate any kind of resource XML file, while second approach is specifically for a layout XML file.

* **Note**:- Above are the steps to **Compare** two files inside Android Studio

### Delete a file in Android Studio

Since we only need one XML file for our purpose, lets delete the renamed file “activity\_send\_message\_m1.xml” from our project.  
**Note: - Never keep unnecessary/unused files in the project, with the project’s growth in no. of source files, it’s easy to lose track of unused files, and they become a burden to the project.**

* There are two ways to delete a file, Delete and Safe Delete

1. **Delete a File**: - Right click on the file to be deleted “activity\_send\_message\_m1.xml” and select “**Delete**” from menu, or simply press “**Delete**” key in keyboard after selecting the file.  
   
2. **Safe Delete a File**: - Right click on the file to be deleted “activity\_send\_message\_m1.xml” and select “**Refactor**”🡪 “**Safe Delete**” from menu, or simply press “**Alt + Delete**” key in keyboard after selecting the file.  
   

**Delete vs Safe Delete**:-   
Delete option will simply delete the selected file, and errors might occur in the code, if that files was being used in the project by other files.

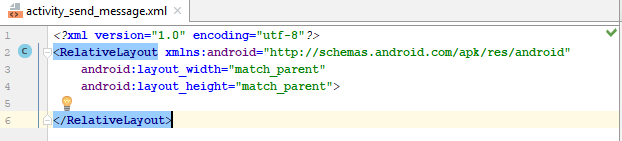
But using Safe Delete, you can safely remove the selected file, keeping the code working and error-free.   
The Safe Delete Refactoring finds all the usages of the selected symbol within the open project or simply delete the symbol if no usages found.

Note: - Always try to use safe delete when and where possible.

**Note**: - Above are the steps to **Delete** a file inside Android Studio

# Design a User interface

Now let’s design a simple user interface by adding (Views) widgets to the layout file “activity\_send\_messag.xml” we created above. Up to now the layout file in its text views looks like below,



Before going forward, it is important to understand the content of the current file.

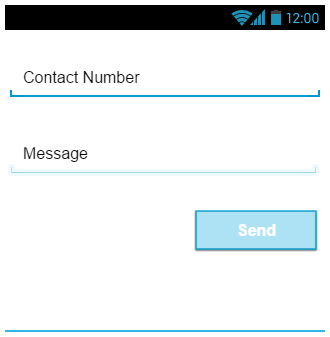
* The First Line:- *<?***xml version="1.0" encoding="utf-8"***?>*
  + This line is called the XML prolog, and not specific to android, but commonly available in any type of XML file
  + It is optional, but if it exists, it must come first in the document.
  + XML documents can contain international characters, like Norwegian øæå or French êèé.
  + To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.
  + UTF-8 is the default character encoding for XML documents.
  + To read more about basics of XML file syntax rules please refer: - <https://www.w3schools.com/xml/xml_syntax.asp>
* The second line:-  
  <**RelativeLayout xmlns:android=**[**http://schemas.android.com/apk/res/android**](http://schemas.android.com/apk/res/android)
  + Every XML file should have a root tag, and above is a part of the root tag of this layout XML file
  + Since we created the layout using “RelativeLayout” as out root tag its has been added to the layout file
  + The second part “xmlns” declares a Namespace as “**android**”
  + The namespace has pretty much the same uses as the package name in a Java application
  + As the value of the “andorid” namespace, a Uniform Resource Indicator(URI) is given
  + In computing, a uniform resource identifier (URI) is a string of characters used to identify a name of a resource.
  + Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.
    - e.g.: http://schemas.android.com/apk/res/android:id is the URI here
* The third and fourth lines (note the usage of defined namespace “**andoird**”):-  
  **android:layout\_width="match\_parent"  
  android:layout\_height="match\_parent"**>
  + **android:layout\_width -** Defines the width for the group; as a dimension value or a keyword
  + **android:layout\_height -** Defines the height for the group; as a dimension value or a keyword
    - The height and width value can be expressed using any of the dimension units supported by Android (px, dp, sp, pt, in, mm)
    - or with the following keywords defined in below table;

|  |  |
| --- | --- |
| **match\_parent** | Sets the dimension to match that of the parent element. Added in API Level 8 to deprecate fill\_parent. |
| **wrap\_content** | Sets the dimension only to the size required to fit the content of this element. |

* Last line is the closing tag of the “RelativeLayout” marking end of the layout

With the knowledge about the current content of the “activity\_send\_messag.xml” file, now its time to design our first simple UI.

The sketch of the UI needs to be designed:-



* Above sketch was designed using “Pencil” software, its free and available at <https://pencil.evolus.vn/Downloads.html>

Steps to implement above design as an Android User Interface:-

# Different Types of Layouts Managers

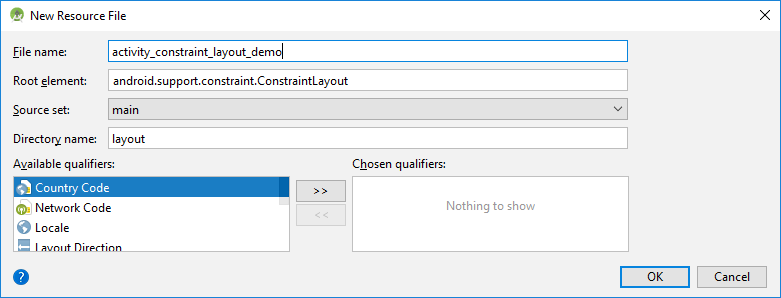
Since we came across the term “RelativeLayout”, it is a good time to learn about different types of Layouts available in android.   
To have a good understanding about Layout types helps to design much better organized UIs, to address the worst limitation when it comes to mobile devices “**Limited Space in UI**”

## ConstraintLayout

ConstraintLayout is provided by an external library.

* + It allows you to use a flat view hierarchy and has great performance.
  + Also the design tools support constraint layout very well.
  + **New projects should prefer the usage of constraint layout.**
  + Android ConstraintLayout is used to define a layout by assigning constraints for every child view/widget relative to other views present.
  + A ConstraintLayout is similar to a RelativeLayout, but with more power.
  + The aim of ConstraintLayout is to improve the performance of the applications by removing the nested views with a flat and flexible design.

Let’s create an example layout to check how to use the ConstraintLayout

1. Add a layout xml file named “activity\_constraint\_layout\_demo”, by adding a “ConstraintLayout” as root element as given below;  
   
2. To the created XML file and a EditText view and a Button View and recreate the UI we designed in above

* There are several attributes in ConstraintLayout to define the size or position of a view.
  + **ratio**: To size elements you can define an aspect ratio (e.g., 16:9).
    - To define an aspect ratio one dimension has to be set to 0dp (match constraints).
    - In xml you can use app:layout\_constraintDimensionRatio.
  + **barriers**: To align elements which size change dynamically you can define a barrier.
  + **chains**: To position multiple elements at once you can define a chain. A chain groups multiple elements.

https://developer.android.com/training/constraint-layout/

## Attributes of UI components

## Usage of string.xml file

## Usage of colours.xml file

## Usage of styles.xml file

# Access and Modify UI from Activity class

# Toasts

# Intents

# References

*Application Fundamentals*. (2018, May 2018). Retrieved from Android Developers: https://developer.android.com

MULLIS, A. (2017, November 11). *Android Studio tutorial for beginners*. Retrieved from Android Authority: https://www.androidauthority.com/android-studio-tutorial-beginners-637572/

SINICKI, A. (2017, December 16). *Android SDK tutorial for beginners*. Retrieved from Android Authority: https://www.androidauthority.com/android-sdk-tutorial-beginners-634376/

Vogel, L. (2016, June 20). *Getting started with Android development - Tutorial*. Retrieved from vogella: http://www.vogella.com/tutorials/Android/article.html#high-level-overview-of-android-development