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

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



Table of Contents

[Introduction 2](#_Toc516169722)

[Android Studio’s UI Design View - Layout Editor 2](#_Toc516169723)

[The Design View of Layout Editor 3](#_Toc516169724)

[Toolbar items to Change the preview appearance 4](#_Toc516169725)

[Edit View attributes 5](#_Toc516169726)

[The Text View of Layout Editor 6](#_Toc516169727)

[Android Views and View Groups 7](#_Toc516169728)

[Layout files 9](#_Toc516169729)

[Create a Layout file 9](#_Toc516169730)

[Rename a file in Android Studio 11](#_Toc516169731)

[Compare two files using Android Studio 13](#_Toc516169732)

[Delete a file in Android Studio 15](#_Toc516169733)

[Design a User interface 16](#_Toc516169734)

[Usage of string.xml file 21](#_Toc516169735)

[Usage of colours.xml file 29](#_Toc516169736)

[Usage of styles.xml file 33](#_Toc516169737)

[Access and Modify UI from Activity class 36](#_Toc516169738)

[How to change application's starting activity 39](#_Toc516169739)

[Accessing views from the layout in an activity 41](#_Toc516169740)

[Navigating between Activities 43](#_Toc516169741)

[Toasts 47](#_Toc516169742)

[Intents 49](#_Toc516169743)

[References 54](#_Toc516169744)

# 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.

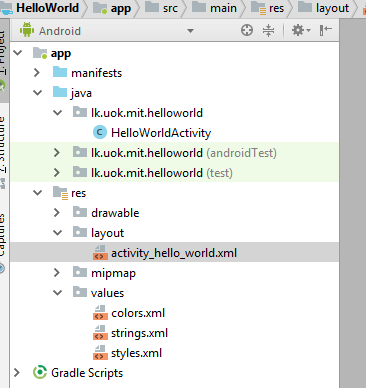
With that knowledge in hand, in this session below areas will be covered,

* 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.

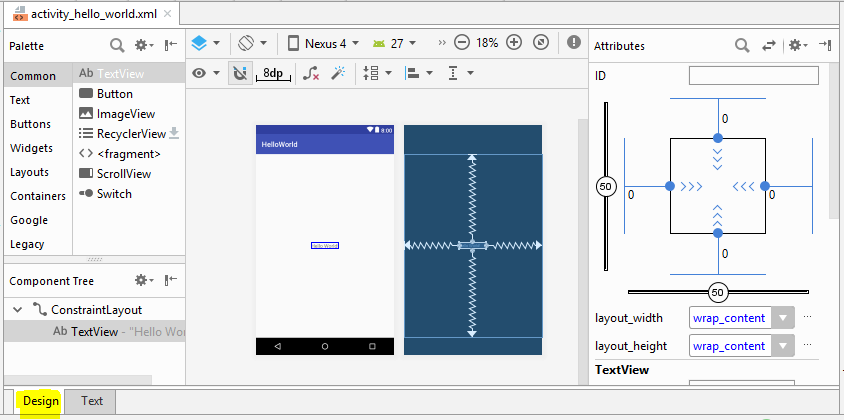
To open the layout editor, a layout xml file should be opened. If you could recall, we added an “Empty Activity” during creation of a new project, and named it as “HelloWorldActivity”. Along with that, an xml file was created as “**activity\_hello\_world.xml**”. This layout xml file could be seen at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

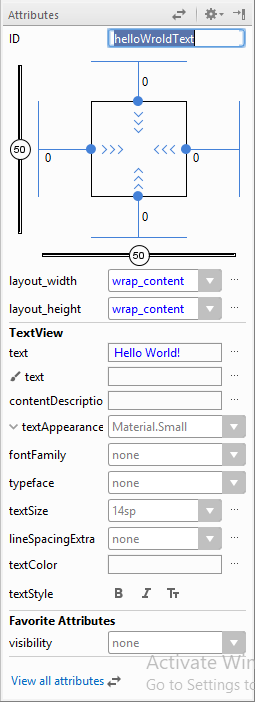
The buttons in the top row of the design editor allow you to configure the appearance of your layout in the editor. This toolbar is also available in the text editor's Preview window.



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

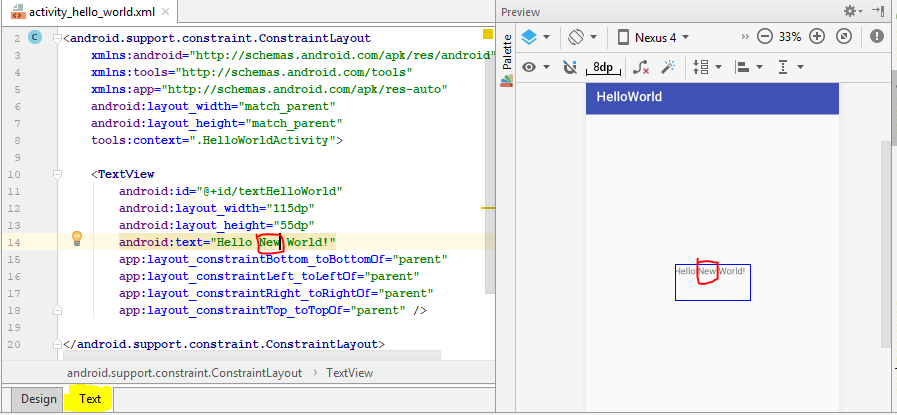
* **Design and blueprint**: Select how you'd like to view your layout in the editor; select either the
  + **Design view** (a real-world preview of the layout),
  + **The Blueprint view** (only outlines for each view),
  + OR Design + Blueprint for both side by side.
* **Screen orientation and layout 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.

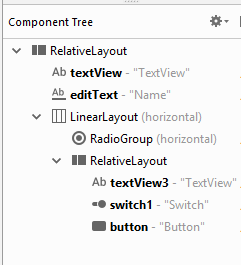
1. Programmatically add View and ViewGroup components (i.e. “widgets”) to the Activity in Java program code.
   * The programmatic approach is typically reserved for more dynamic and/or complicated UI scenarios.
2. Use XML layout files.
   * The typical and most common way to construct an Activity.

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

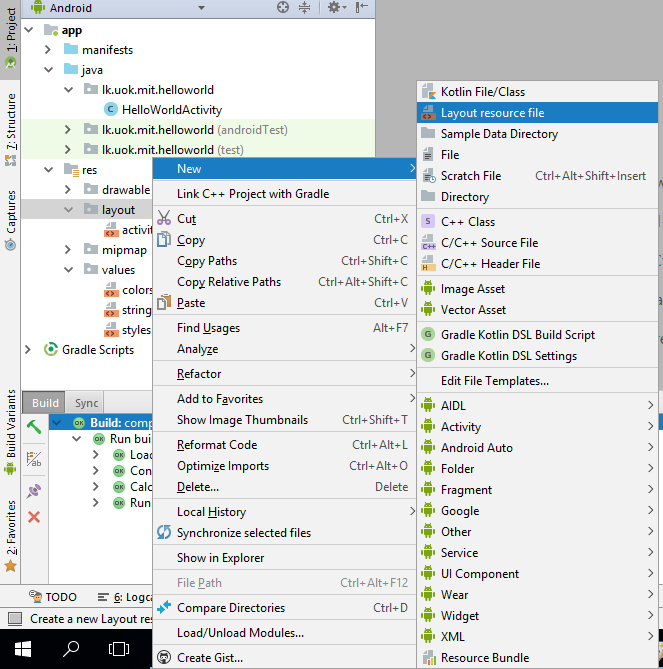
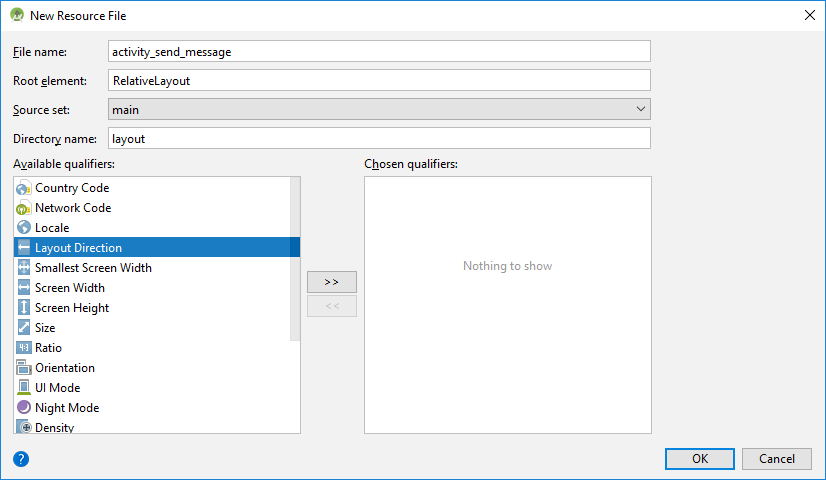
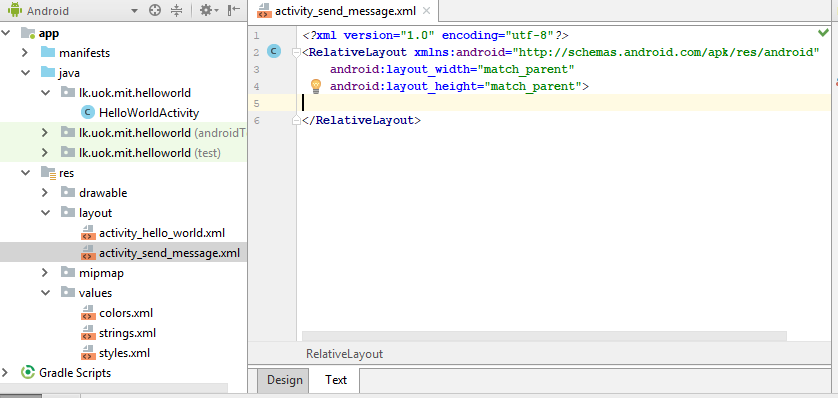
via an XML representation.

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!**

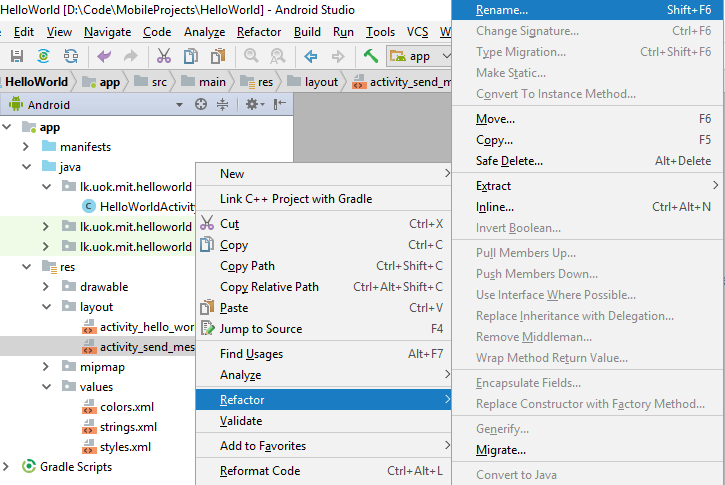
#### 1st way: - Using New🡪 “Layout resource file” option

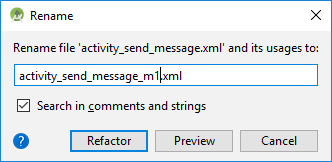
1. Step 1:- Got to **res/layout folder**, **right click** on it, select “**New**”🡪 “**Layout resource File**” as shown below,  
   
2. Step 2:- After above step, below screen will appear,  
   
3. Step 3:- Fill in the details in the dialog:
   1. **File name**: Type the name for the XML file (does not 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

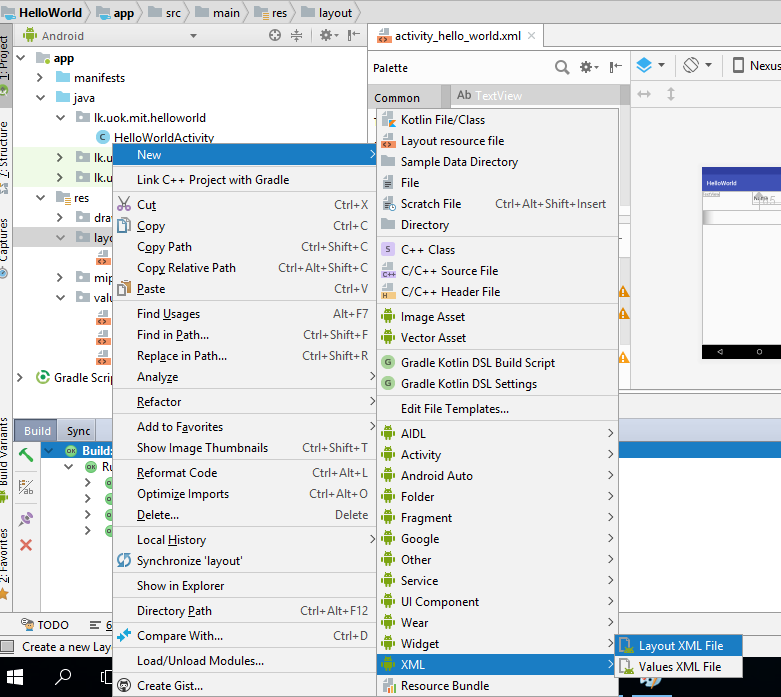
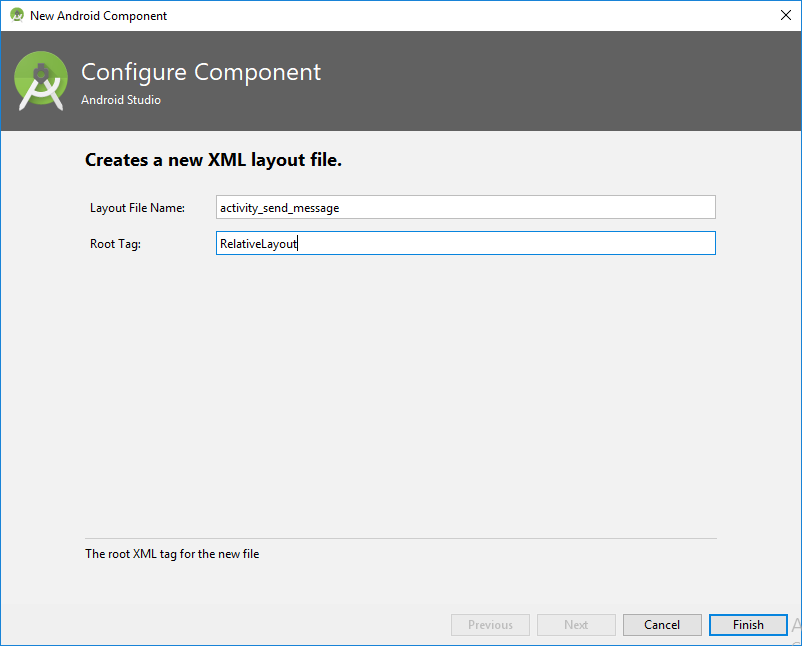
Rename the “activity\_send\_message.xml” file generated from above to “activity\_send\_message\_m1.xml” like shown below, this will be deleted later as we need only one xml file;

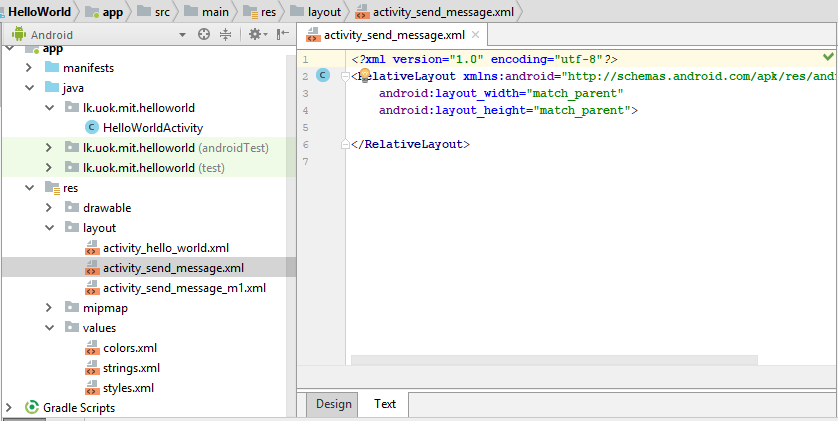
1. Step 1:- Go to **res/layout folder**, **right click** on “**activity\_send\_message.xml**”, select **Refactor**🡪”**Rename**” as shown, (or Shift + F6)



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



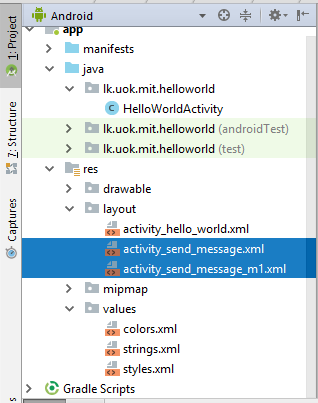
### Compare two files using Android Studio

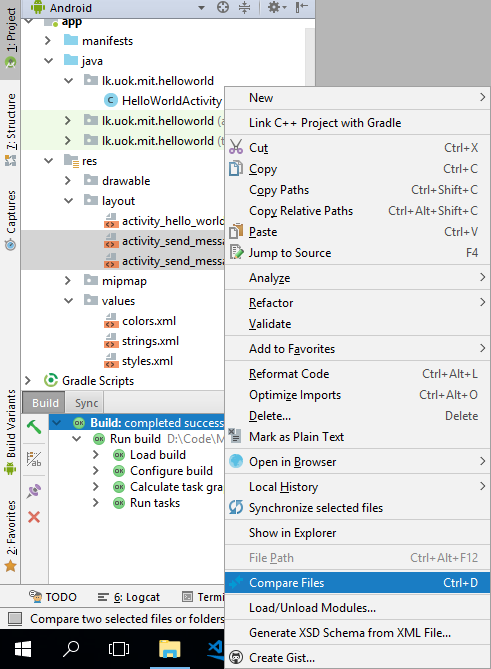
Above it is shown how to create a layout xml file with “RelativeLayout” as its root element, using two different methods.

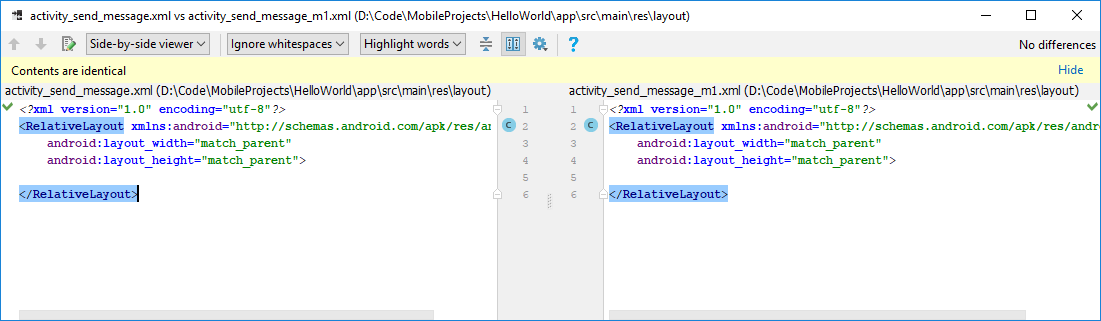
Now you should have below two files inside your **res/layout** folder as shown in above figure

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  
   

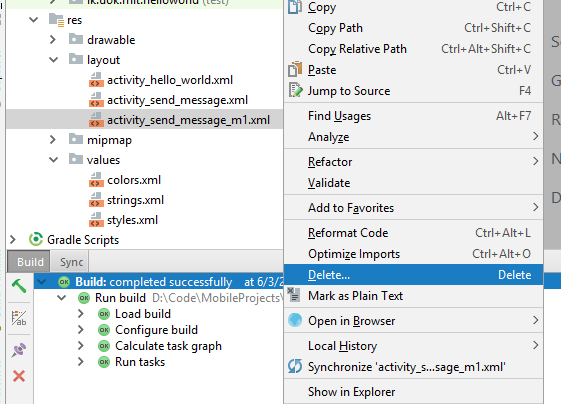
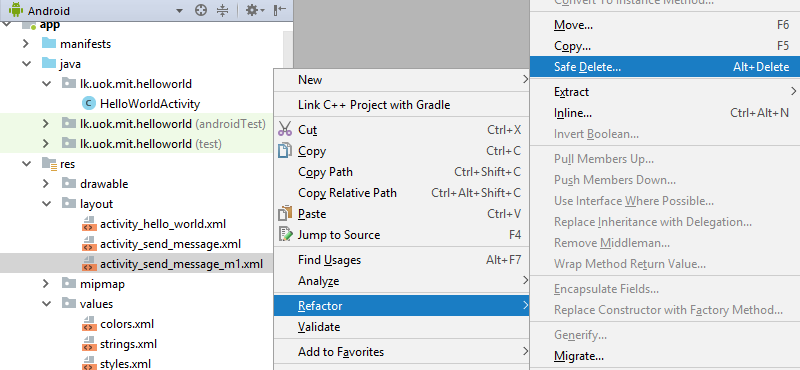
* As it is shown in above image, there are no differences between the two files.
* Therefore we can conclude that from both approaches mentioned above, a similar layout file could be generated.
* To add a further note here, first approach can be used to generate any kind of resource XML file, while second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

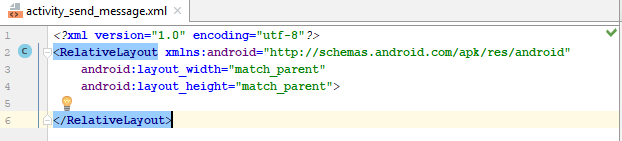
But using Safe Delete, you can safely remove the selected file, keeping the code working and error-free.   
The Safe Delete Refactoring finds all the usages of the selected symbol within the open project or simply delete the symbol if no usages found.

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.

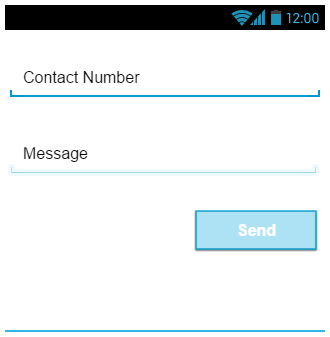
* The First Line:- *<?***xml version="1.0" encoding="utf-8"***?>*
  + This line is called the XML prolog, and not specific to android, but commonly available in any type of XML file
  + It is optional, but if it exists, it must come first in the document.
  + XML documents can contain international characters, 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

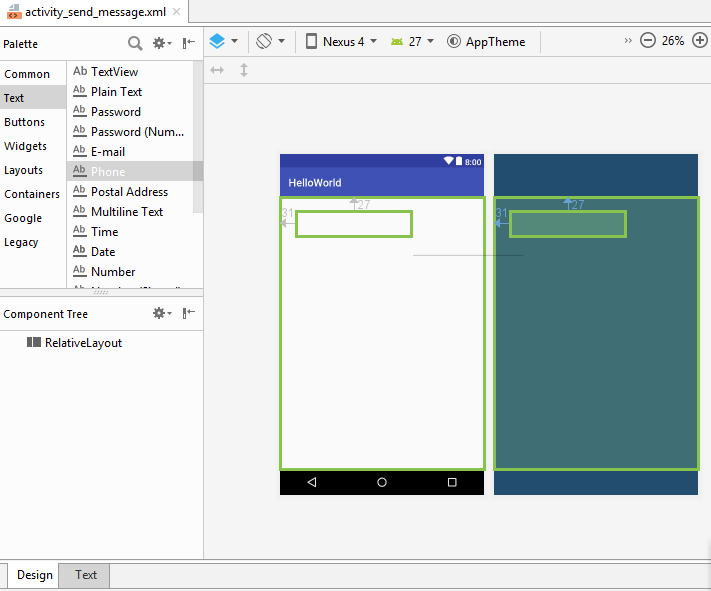
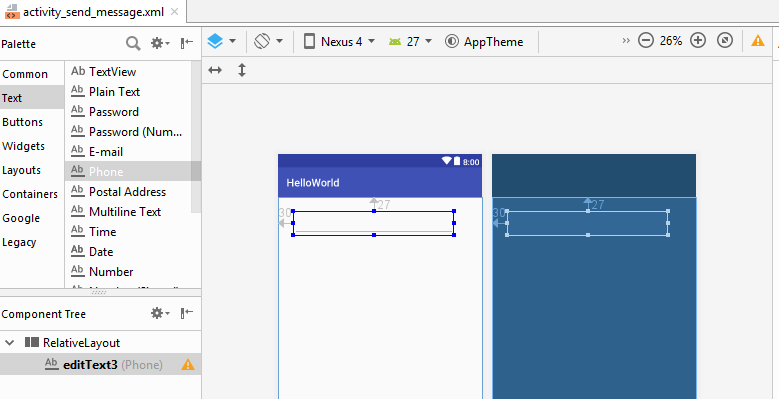
With the knowledge about the current content of the “activity\_send\_messag.xml” file, now its time to design our first simple UI.

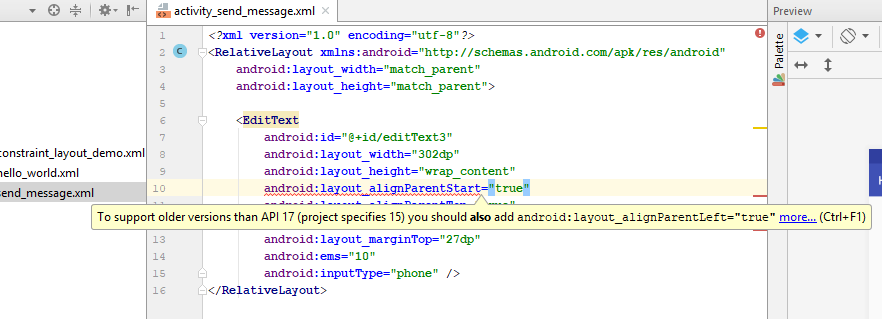
The sketch of the UI needs to be designed and the expected android elements and specific attributes:-



* Contact number **EditText**:- **Pending**  
  id:- **editTextContactNumber**  
  inputType:-**phone**hint:- Contact Number
* Message **EditText**:- **Pending**  
  id:- **editTextMessage**inputType**:-text|textLongMessage**hint:- Message
* Send **Button**:- **Pending**  
  id:- buttonSend
* 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:-

1. Open the layout file “activity\_send\_messag.xml” in android studio
2. Select “Design” view if it’s not already selected
3. Select “Text” item from “Palette” window
4. Drag “Phone” EditText view on to the layout design and drop on layout design  
   
5. Now reposition and resize (as shown below) it using the cursor till it looks like the sketch given above  
   
6. Note the circled icon in above figure, it indicated there are errors/warnings in the layout design
7. Ti fix the errors
   1. Select “Text” View in the window, it will look like below

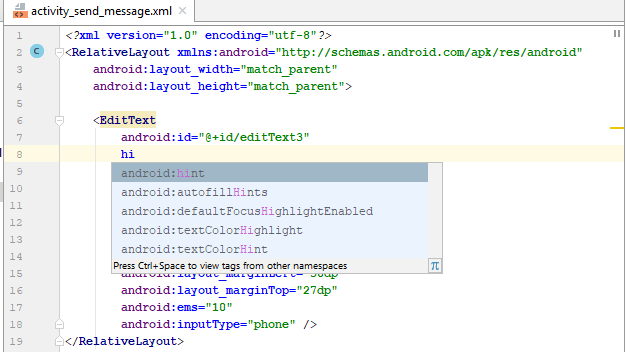


* 1. Please note that the attributes like “android:layout\_width” will be different based on the positioning and resizing done by you
  2. To fix the layout issues, first we should identify the cause of the error
  3. As shown above, **a yellow color tooltip** will appear when “**hover**” on the line marked in red color, explaining the cause of the error.
  4. The cause of the error shown above is, as shown in the tooltip, the attribute “**android:layout\_alignParentLeft="true"” should be added** in to the EditText, since the minimum supported android version in marked as 15 in the project and this attribute is mandatory in such older versions.
  5. The second error in the EditText view was like below, here the link on the tooltip “more” has been clicked, then it shows more details about the error.

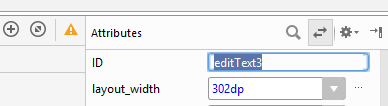
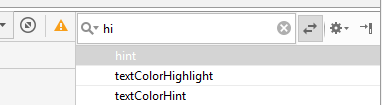
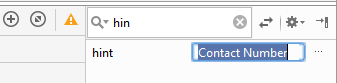


* 1. To fix the above error, an attribute should be added to the EditText as “android:layout\_marginLeft="30dp"”, since older devices would recognize only that attribute, not **marginStart**
  2. **Now all the shown errors are fixed,** and the layout file text view will look like below; Note the yellow color line  
     ****
  3. Even all the errors are gone, there Is a warning left, to view details about the warning “hover” the cursor over the yellow line circled in above figure.
  4. As the tooltip indicates, add the missing attribute “android:hint” to fix the warning.

1. To add an attribute using text view
   1. Start editing the XML file by pressing “Enter” key on the line you want to enter the attribute
   2. Start typing the required attribute name “eg:- hi..” (to enter the attribute named android:hint)
   3. Then the IDE will list down the suggestions as shown in below,
      1. **To view ALL the available attributes, simply press “ctrl + space” keys in the desired location, then to filter, start typing the desired attribute name.**



* 1. Once the required attribute appears in list, simply click on it

1. To add an attribute using “Attributes” view, first go to “Design” view, and them select “View all Attributes” in “Attributes” view  
     
   1. Then scroll down to the desired attribute/ or type attribute’s name on **search field** on top of “Attributes” window  
      
   2. Once the desired attribute appears on search click on “View All Attributes” mark again to go back to default view of “Attributes” window, like shown below  
      
   3. Here, to add a value for the hint attribute, there are two options
      1. Enter a static text as shown above (a value like “Contact Number”)
      2. Enter a value from string.xml file, by clicking on button next to text box above (**this is the correct usage**)

## Usage of string.xml file

In the last step above, it was mentioned that a static text value can be replaced by a value from string.xml file. When adding your strings to strings.xml, you can easily translate your whole app into other languages.

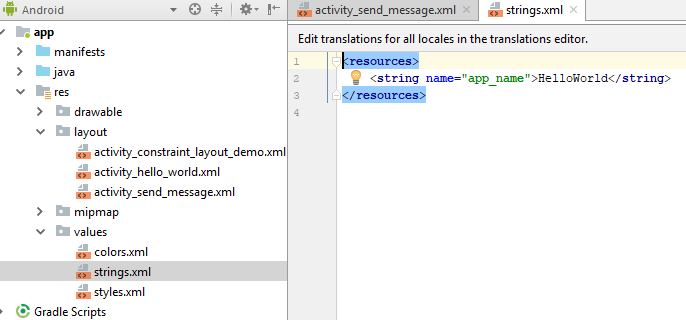
Example:

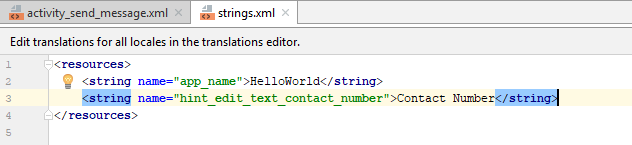
Suppose that your app's default language is English. Suppose also that you want to localize all the text in your app to French, and most of the text in your app (everything except the app's title) to Japanese.   
In this case, you could create three alternative strings.xml files, each stored in a locale-specific resource directory:

* **res/values**/strings.xml
  + Contains English text for all the strings that the app uses, including text for a string named title.
* **res/values-fr**/strings.xml
  + Contain French text for all the strings, including title.
* **res/values-ja**/strings.xml
  + Contain Japanese text for all the strings except title.

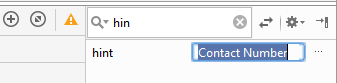
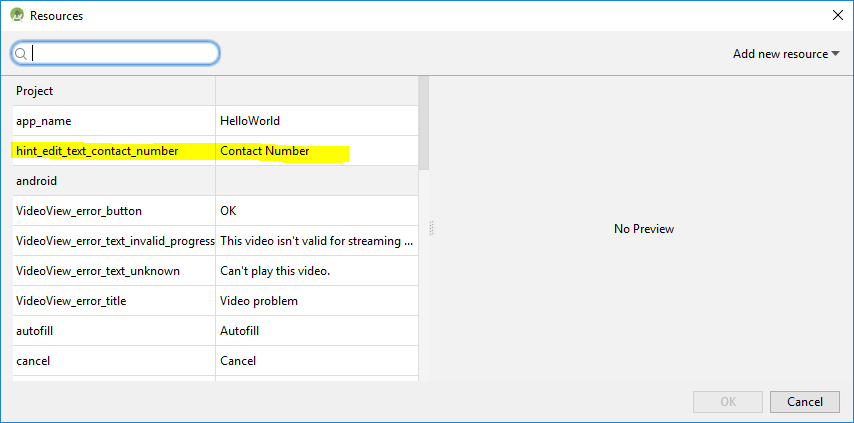
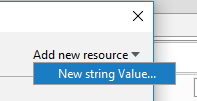
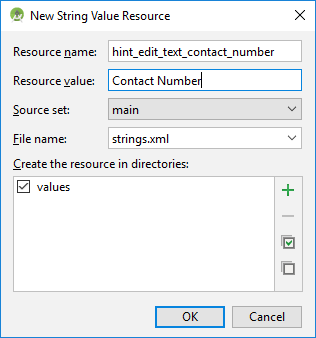
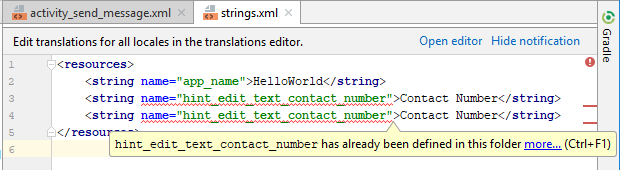
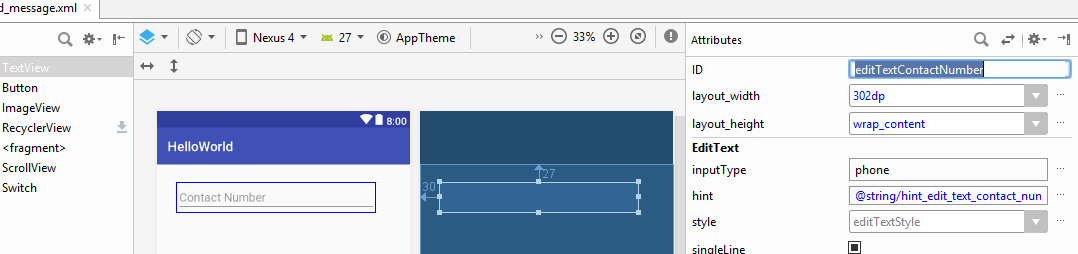
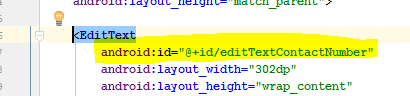
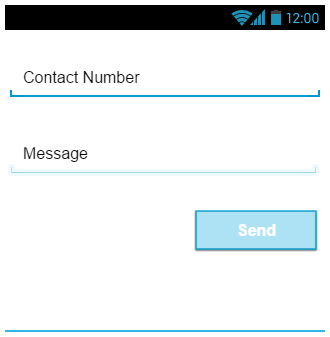
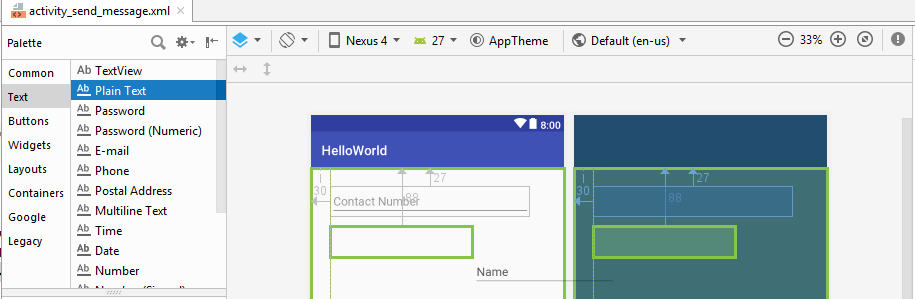
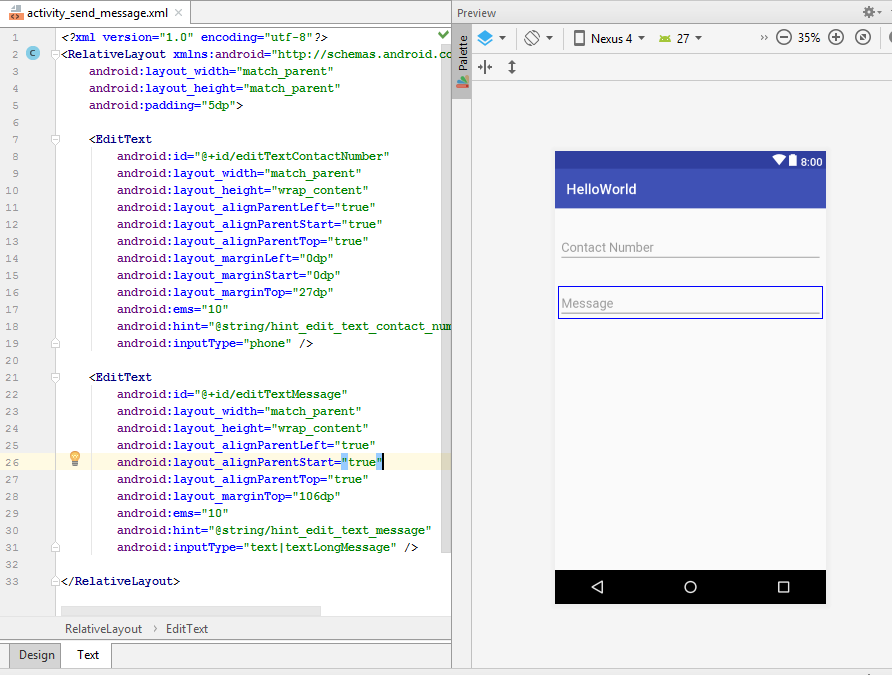
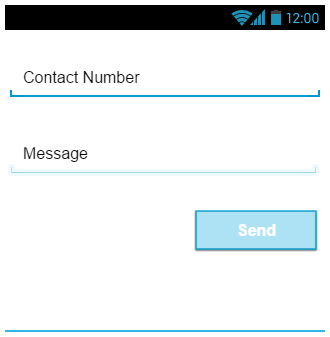
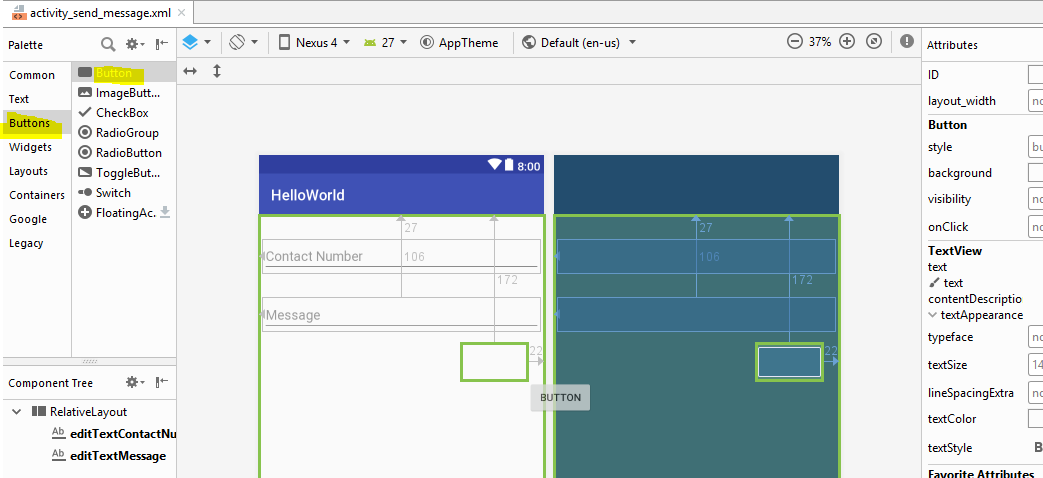
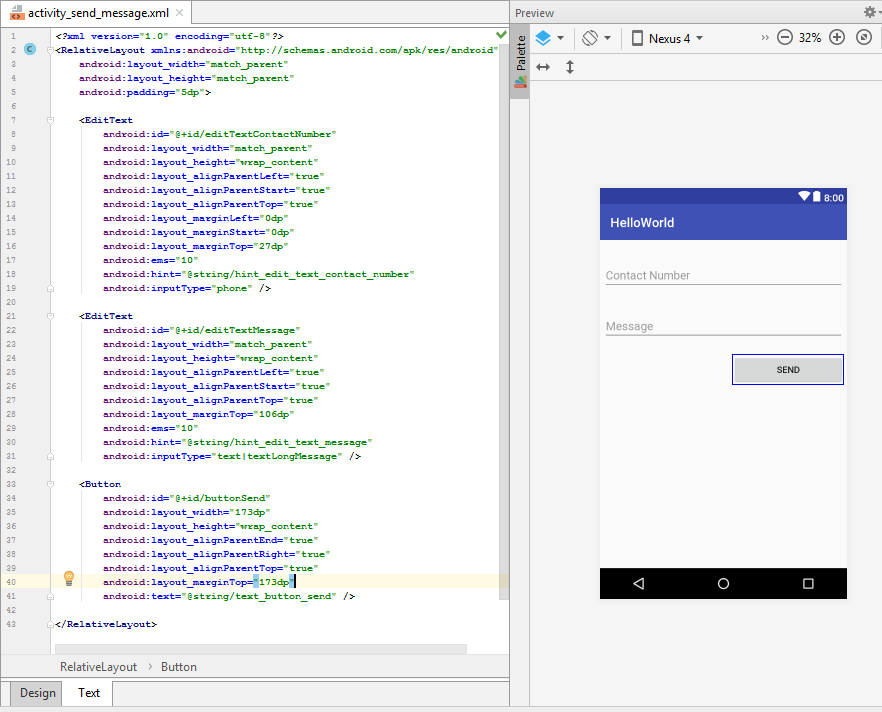
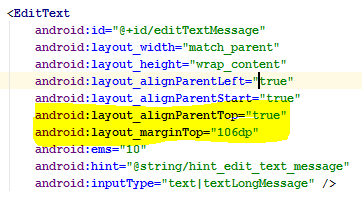
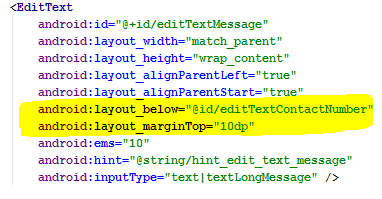
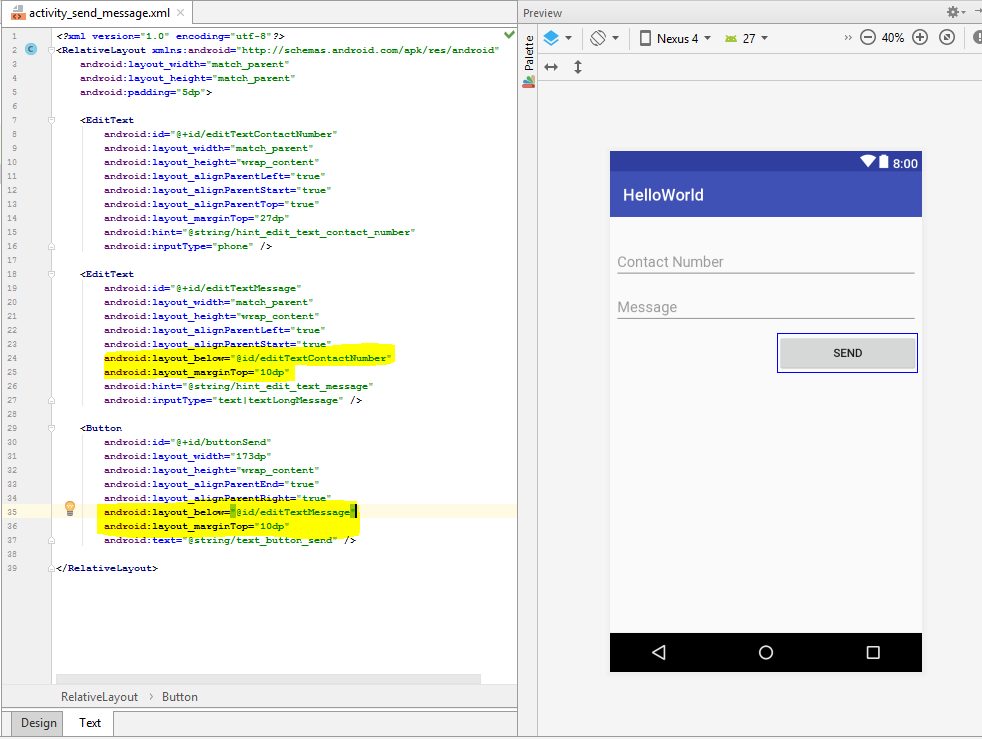
Now let’s see how to use strings.xml in this application.

First open the **strings.xml** file in the application, under “res/values” and check the current content. Should look like below;



* All the strings have been defined using a <string> tag with in two <resources> tags
* The <string> tag has a “**name**” attribute
  + The value of “name” attribute must be unique for the file
  + When using a strings.xml file for a different language, a string with same “name” attribute should be available in that file also.
* Inside the opening and closing <string> tags, the string value is defined
  + Eg:- **<string name="hint\_edit\_text\_contact\_number">Contact Number</string>**
* We can directly edit the strings.xml file and enter required string here
  + Now **Enter the above line in strings.xml file**
* Later can refer to resource in string as:
  + In Java: R.string.string\_name (eg:- **R.string. hint\_edit\_text\_contact\_number**)
  + In XML:@string/string\_name (eg:- :@string/ **hint\_edit\_text\_contact\_number**)
* Now the strings.xml file should look like below;  
  

**Now continuing from last step of the interface design given above**,

* Open the layout file “activity\_send\_messag.xml” again
* Go to Attribute window and search for “hint” attribute and select it as mentioned above
* Click on the button next to the editable filed as shown below;
* Below windows will appear, listing down available resources to be used as a text in (**key-value**) pairs;
* **key- name attribute’s value** and **value – actual text value inside <string> tag**  
  
* Note the highlighted line, it’s the value we entered in strings.xml file
  + if it’s not visible, search for it using either “name” attribute (the key) or the actual text (the value)
  + We can simply click on it and click “OK” to add it as string value of the “hint” attribute
* It is possible to add a new string resource through this UI as well, using “Add new resource” button, click on “New string Value” as shown below  
  
* Then below pop up window will appear, note that exact same values from our previous entry was added in this UI to make it easy to understand  
  
  + Resource name :- the name attribute of the new string resource
  + Resource value:- the actual text value required
  + Source set:- if its test or main resources
  + File name :- to which file the entry should be added to
  + Directory list:- if there are multiple values directories for multiple languages, all will appear here
* Now click on “OK”
* Now open the strings.xml inside “res/values” directory and check its content; should appear like below  
  
* Note the new entry and errors indicated in the file
  + as mentioned above the name of a string resource entry cannot be duplicated
  + try changing the “name” attribute’s value and see errors gone
* However, we don’t need the second entry and therefore simply delete it from strings.xml file
* Go back to our layout file “activity\_send\_messag.xml” and go to text view and observe the xml code part shown below;
  + **android:hint="@string/hint\_edit\_text\_contact\_number"**
    - **The text for hint has been taken as a string resource**
* Add an “id” to the attribute “android:id” of the editText as shown below, using attribute window (can do the same by editing XML as well)  
  
  + Note the value added for the attribute ”ID” above as “**editTextContactNumber**”
  + There are no rules for values used as attributes, but as a convention it is recommended to use the control’s type (editText) and control’s purpose (to hold contact number), then it will be easy to identify this controller by its id later
* Note the XML code in text view, and note the id attribute as shown below;  
  
  + The + sign is a short cut to add the id to your list of resource ids.
  + However, just as the string.xml, we can define id s in a separate xml and use it here if required (then the + sign before “id” is not required)
  + E.g.:- The difference between below two lines;
    - <listview id="@+id/android:list">
    - <listview id="@android:id/list">
  + @+id/foo means you are creating an id named foo in the namespace of your application.
    - You can refer to it using @id/foo.
  + @android:id/foo means you are referring to an id defined in the android namespace.
  + The '+' means to create the symbol if it doesn't already exist.
  + **You don't need it (and shouldn't use it) when referencing android: symbols, because those are already defined for you by the platform and you can't make your own in that namespace anyway.**
* Now it’s time to add the reaming EditText control and the Button in to the UI to complete the UI design  
  
* Contact number **EditText**:- **Added**  
  id:- **editTextContactNumber**  
  inputType:-**phone**hint:- Contact Number
* Message **EditText**:- **Pending**  
  id:- **editTextMessage**inputType**:-text|textLongMessage**hint:- Message
* Send **Button**:- **Pending**  
  id:- buttonSend
* Now the Contact Number is successfully added to the layout file.
* Add another EditText for “Message” component of the sample UI
  + As shown in below image, select a “**Plain Text**” from “Text” category and drag on to the layout file’s “Design” view and drop  
    
  + Position the new EditText below “**editTextContactNumber**”
  + Resize the new EditText to match with “**editTextContactNumber**”’s width.
  + Go to “Text” view of the layout and correct if there are any errors as described in above section
  + Change the attributes of the new “EditText” as shown below,
    - android:id:- editTextMessage
    - android:hint:- Message (make sure to use string resource to define the value)
* After adding the new EditText with id “editTextMessage”, the “activity\_send\_messag.xml” layout’s text view will look like below;  
  
* Tip for UI design:- note the “android:padding” attribute of <RelativeLayout>.
  + The easiest way to resize the UI components is, resize them to full width of parent
  + Then set the padding in parent layout
* Now it’s time to add the reaming Button control in to the UI to complete the UI design – below is the current status  
  
* Contact number **EditText**:- **Added**  
  id:- **editTextContactNumber**  
  inputType:-**phone**hint:- Contact Number
* Message **EditText**:- **Added**  
  id:- **editTextMessage**inputType**:-text|textLongMessage**hint:- Message
* Send **Button**:- **Pending**  
  id:- buttonSend
* As shown in below image, select a “**Button**” from “Buttons” category and drag on to the layout file’s “Design” view and drop  
  
  + Position the new **Button** below “**editTextMessage**”
  + Resize the new Button to match UI design.
  + Go to “Text” view of the layout and correct if there are any errors **as described in above section**
  + Change the attributes of the new “**Button**” as shown below,
    - android:id:- buttonSend
    - android:text:- Send (make sure to use string resource to define the value)
* After adding the new Button with id “**buttonSend**”, the “activity\_send\_messag.xml” layout’s text view will look like below;  
  
  + Notice that there are two EditText components and One Button Component is available the layout “activity\_send\_message.xml”
  + Considering the “Preview” the UI we designed, closely resembles to the required UI
* Some Tips when positioning UI elements;
  + If you could notice, the order we wanted the UI components are,
    - **Contact Number** Input – at the top
    - **Message** Input below “**Contact Number**”
    - **Send Button** below “**Message**”
  + Below is the current XML code for “**Message**” input  
    
    - Note the highlighted properties
    - Even the “Message” is aligned to parent layout top and a margin has been set from parent layout
    - Rather than doing this, it’s easy to align the element against the immediate neighbor element and set margin relative to that.
    - The highlighted attributes above could be replaced from the highlighted properties below;  
      
* Likewise, arrange UI elements relative to the neighboring elements
* The layout XML code will look like below;  
  

## Usage of colours.xml file

To see the usage of colours.xml file, let’s try to change the color of the “Send” button

Android uses standard RGB (red, green and blue) color model. Each primary color value is usually represented by hexadecimal number. At the beginning of such a color definition you have to put a pound character (#).

* The simplest is just #RGB format, where #000 is black and #FFF is white.
* But in this format we have only 16 values per color so it gives 4096 combinations.
* That’s why #RRGGBB format is mainly used.
* In this format we have 256 values per primary color, so 16 777 216 colors in total.
* Black is #000000 and white is #FFFFFF.

We could use also alpha channel information that defines transparency (opacity). There are two formats available

* #ARGB
* #AARRGGBB.

The alpha value 00 means that color is fully transparent (so invisible) and FF that it’s not transparent at all (opaque), so you could skip this information.

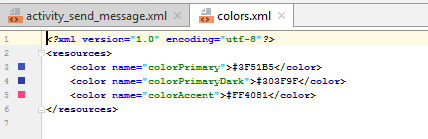
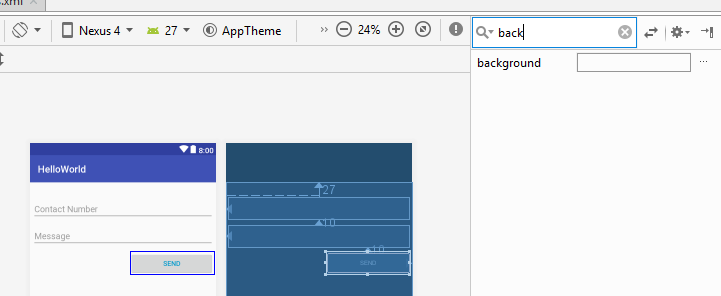
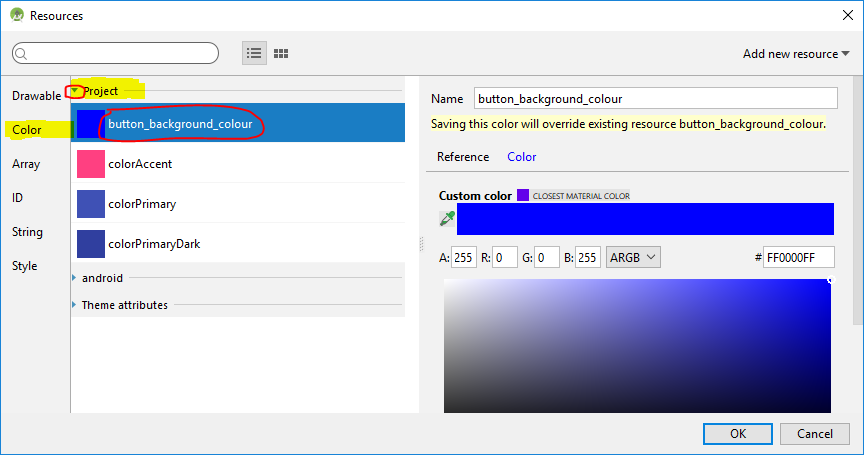
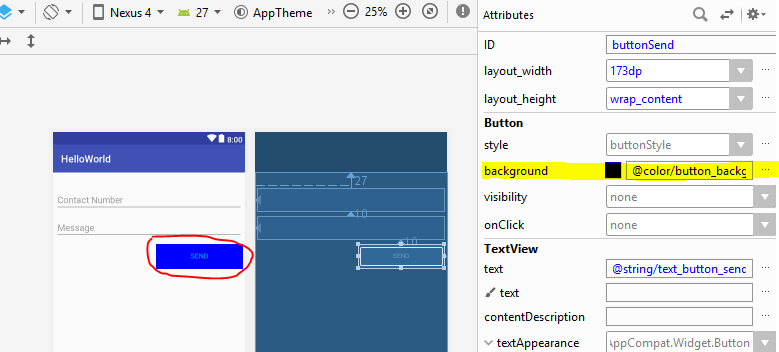
There are various options how to set colors in Android app. Such as putting the color definition directly into proper XML tag attribute or in Java Code, for instance:

* **android:textColor="#AARRGGBB" -**  using hexadecimal color code
* **android:textColor="@android:color/holo\_blue\_dark" -** using defined colors in android

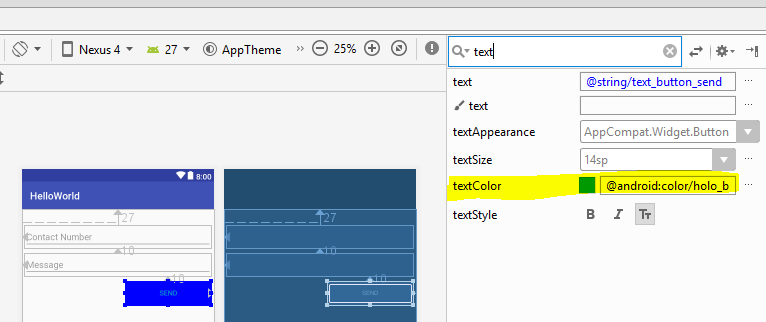
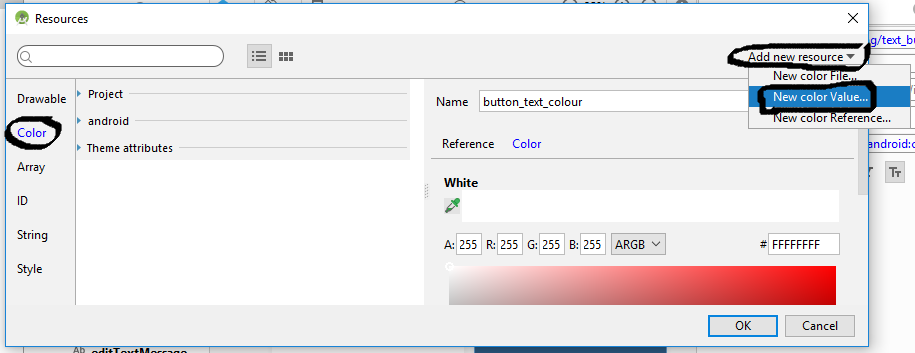
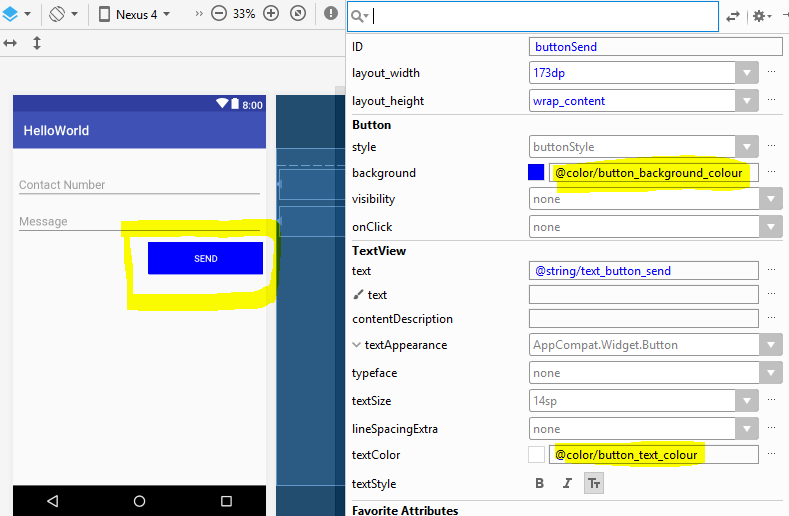
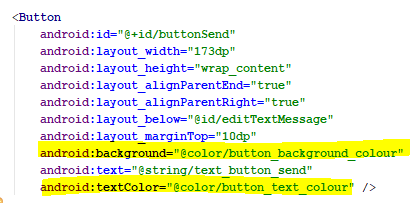
The best solutions is to define your **own colors** set in the **colors.xml** file.

* It’s much easier to remember natural names than hex values.
* In colour.xml you could use names not only referring to direct colors like “dark\_blue”, but use also various other names like “menu\_color” or “color\_of\_introduction”.
* You could easily change color for all app modifying one line of code and you could even have different colors for smartphones and tablets.
* The colors defined in colors.xml could be used both in XML and in Java.

**Lets define a color as “button\_background\_colour” and use it for the “Send” button**

1. Open the colors.xml file under res/values folder  
   
   1. The content of the file will look like above, there are three colors defined already upon project generation
   2. To add a new colour definition with name as “**button\_background\_colour**” and colour as “blue” the entry will look like below, Note:- the colour code for blue is **#0000ff  
      <color name="button\_background\_colour">#0000FF</color>**
2. Open the “**activity\_send\_message.xml**” file and go to design view
3. Select the “**buttonSend**” and search for the attribute “background” and click on button next to text box  
   
4. The below pop up box will appear after above step, and from that, select the “Color” section, and under that, expand “Project” section, and you will be able to see the color we added to the colours.xml file as shown below;  
   
   1. Note that, the defined colour can be modified using a colour picker here.
   2. After selecting the “**button\_background\_colour**”, click “OK”
   3. Note :- The “Add new resource” option in this window could be used to enter a new entry to the colours.xml file
5. Now the “**activity\_send\_message.xml**” will look like below, in the design view  
   
6. As you could see in the above image, after changing the background colour of the Button, it is hard to see the text on it

**Let’s change the colour of the text of “buttonSend” using “Add new Resource” option as mentioned in step 5 above**

1. Open the “**activity\_send\_message.xml**” file and go to design view
2. Select the “**buttonSend**” and search for the attribute “textColor” and click on button next to text box  
   
3. In the popup window appear as shown below, select “Color” section  
   ****
   1. Then go to “Add New Resource” and click on it
   2. From the drop down, select “New color Value”
   3. In the section appear, enter the name of the color as “**button\_text\_colour**”
   4. Then select white using color picker or enter as “#FFFFFFF” in the text box
   5. Then click “OK”
4. Now the “Design” view of “**activity\_send\_message.xml**” will appear like below  
   
5. Now go to the “Text” view and check the XML code for “buttonSend” button, it will look like below  
   

## Usage of styles.xml file

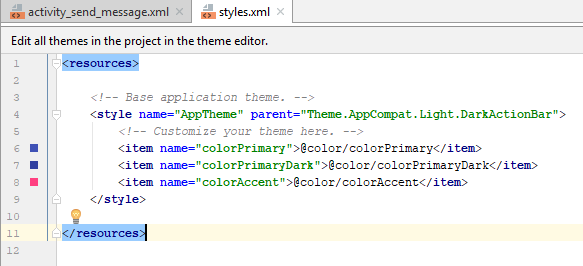
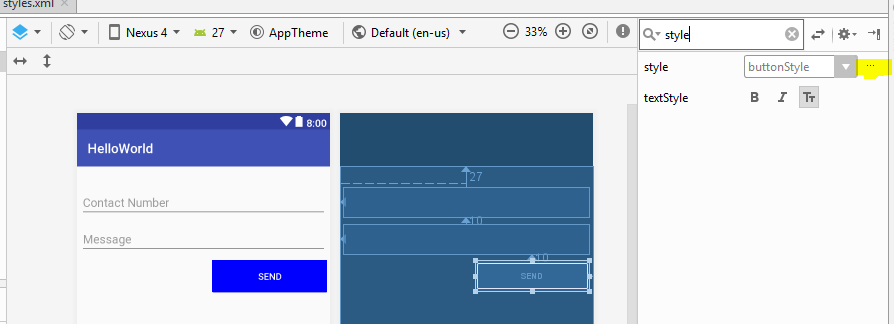
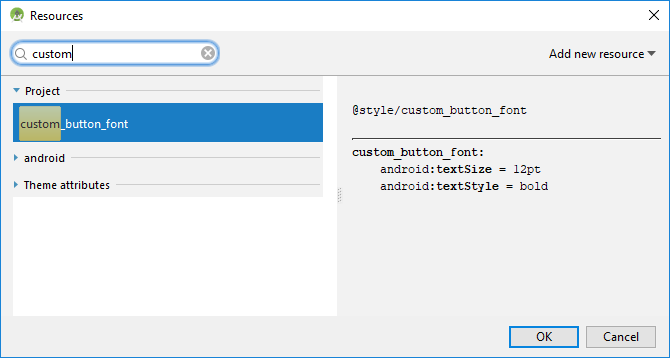
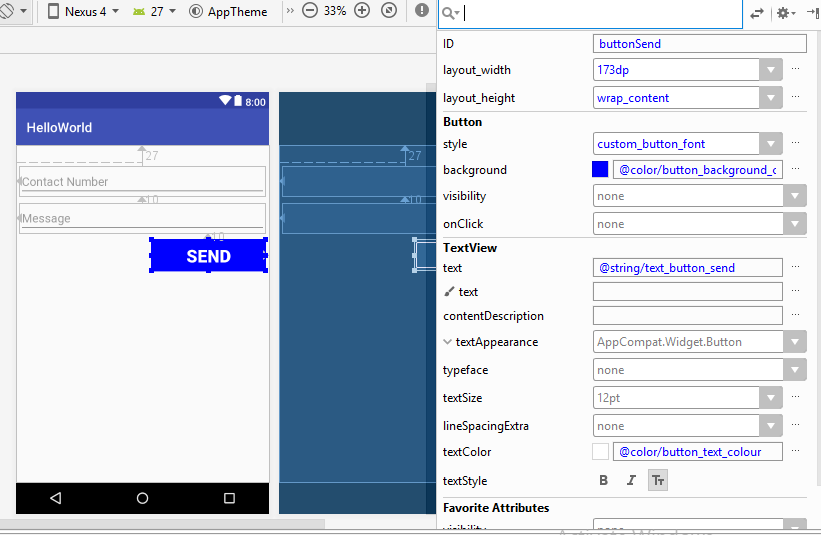
Styles and themes on Android allow you to separate the details of your app design from the UI structure and behavior, similar to **stylesheets** in web design.

* A style is a collection of attributes that specify the appearance for a single View.
* A style can specify attributes such as font color, font size, background color, and much more.

A theme is a type of style that's applied to an entire app, activity, or view hierarchy, not just an individual view. When you apply your style as a theme, every view in the app or activity applies each style attribute that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Styles and themes are declared in a style resource file in res/values/, usually named **styles.xml**.

Let’s create and apply a style for the “buttonSend” button to make its text bold and text size to 12

1. Open the “styles.xml” file inside “res/value” folder
2. The current content of the file will look like below;  
   
   1. A style named “AppTheme”has been specified and as you could see, colour resources defined inside “colors.xml” also have been used here
   2. The “AppTheme” is an example of inheriting a style
   3. Android supports style Inheritance in very much similar way as cascading style sheet in web design.
   4. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add (Just like in “AppTheme”)
3. You can define multiple styles per file using <style> tag but each style will have its name that uniquely identifies the style.
4. Android style attributes are set using <item> tag
5. To define a style to **make text bold and text size to 12**, a style should be defined like below,<**style name="custom\_button\_font"**>  
    <**item name="android:textSize"**>12pt</**item**>  
    <**item name="android:textStyle"**>bold</**item**>  
   </**style**>
   1. The style name is given as “custom\_button\_font”
   2. For that style, two items have been added to override the attributes requested above
6. Next, to apply this style to the “**buttonSend**”,
   1. Open “**activity\_send\_message.xml**” file and go to design view
   2. In “Attributes” search for the attribute “style”  
      
   3. After the step above, below window will appear, search for the new style name we added, and click “OK”  
      
7. Now go to the “Design” view of “**activity\_send\_message.xml**” and see the “**buttonSend**”, it will look like below;  
   
8. Check the XML for “buttonSend”, it will look like below;  
   

With this, the basic UI designing part comes to an end. Up to now, we looked in to

* Android studio’s layout editor
* Android studio functionality
  + How to Rename a File
  + How to Compare Two Files
  + How to Delete a File
* Different Types of View
* How to add a Layout XML file
* Positioning and Resizing Views
* Constructing UIs using components in pallet
* Changing attributes of different view
  + By Editing XML
  + By using “Attribute” window
* Usage of strings.xml file
  + how to add a string resource
  + how to use a string resource in a layout
* Usage of colors.xml file
  + how to add a color resource
  + how to use a color resource in a layout
* Usage of styles.xml file
  + how to add a style resource
  + how to use a style resource in a layout

With this knowledge in hand, lets move in to next part and check how we could access the layout we designed from a Java Activity class

# Access and Modify UI from Activity class

Now we have created a layout name “**activity\_send\_message.xml**”, and completed the UI design of it. Our next task is to load this view when running the android app.

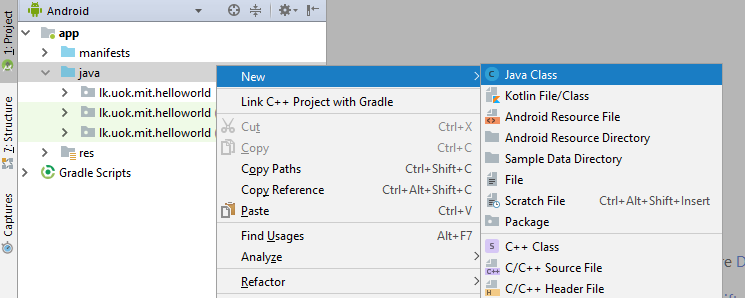
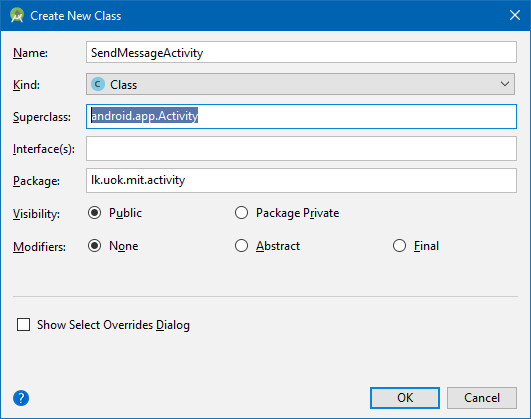
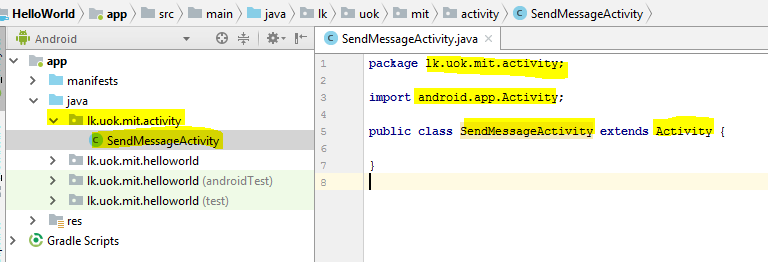
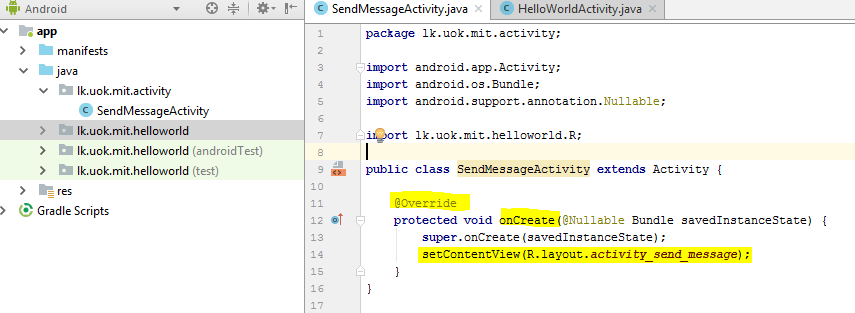
As it was mentioned in the tutorial part 1, in order to create an android UI, two components should be combined, that is

* an Activity class
* a layout XML file

Since we already have the XML, let’s create an Activity class, as shown in the steps below;

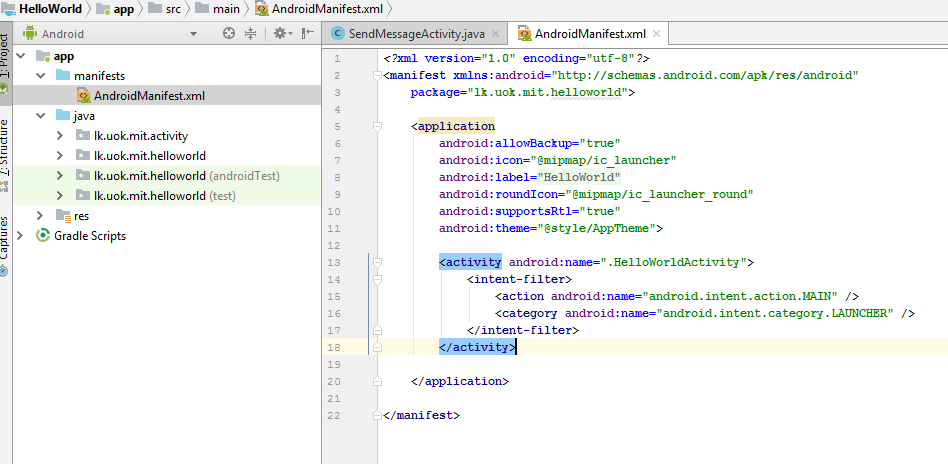
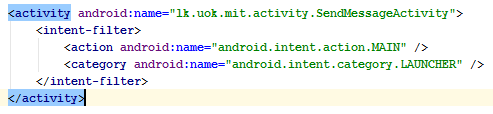
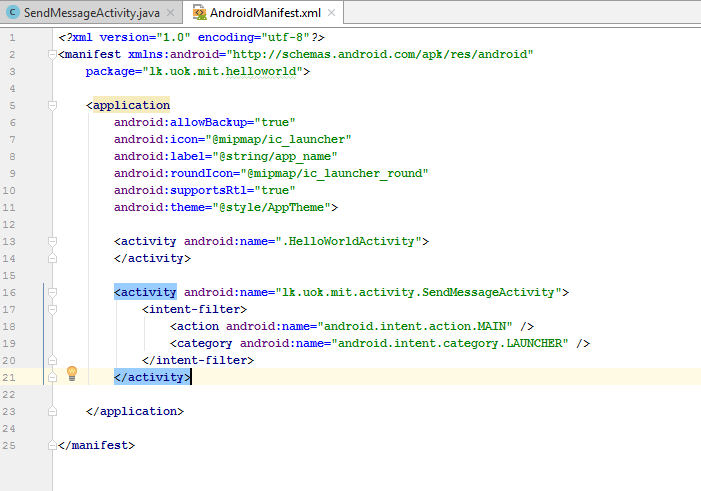
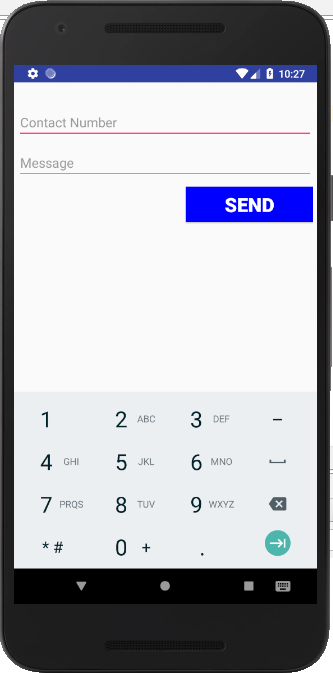
Activity is a Java class, all the Java classes should be created under “java” folder. Naming convention in Java applies here also. All the java classes should be created under a specific package.

In order to get a good understanding about an activity class, lets create a normal Java class and convert it in to an activity class by editing code.

1. In the “Project” view, right click on “Java” folder and got to “**New**”-->”**Java Class**”  
   
2. After the step 1, below window will appear, fill in the fields as shown here;  
   
   1. **Name**:- This is the name of Java class
   2. **Kind**:- Types of the item created (e.g.:- Class, Interface, Enum, etc.), We need a “**Class**”
   3. **Superclass:-** If this class should be extended from a superclass, mention it here with full qualified name- in our case, since we are creating an activity, it should be “**android.app.Activity**”
   4. **Interfaces :-** If the created class inherits from an interface, it should come here, as same as Superclass, the full qualified name is required
   5. **Package :-** The name of the package of our class, note that there is no package in our project as “**lk.uok.mit.activity**”, but it will get created along with the class
   6. **Visibility :-** The access modifier of the class:- should be **“Public”**
   7. **Modifiers :-** If the class should be marked as Abstract or Final, we should select “**None**”
   8. After filling above information, click on “OK”
3. The generated class will look like below;  
   
4. After Step 3, we have an activity class named “**SendMessageActivity**” and a layout XML file named “**activity\_send\_message.xml**”.
   1. To get a beter understanding, think of the **activity class** as the “**Controller**” and the **layout XML** file as the “**View**”
   2. **In this step, let’s see how to make the controller aware of its view**
   3. Below code snippet is responsible for binding the layout xml file (**activity\_send\_message.xml**) to the Activity class “**SendMessageActivity**”   
        
      **setContentView(R.layout.activity\_send\_message);**
      1. setContentView method is inherited to our activity class from the superclass “Activity”
      2. It is a self-descriptive name, the purpose is to set the content view
      3. As the parameter, we should give a layout xml file
      4. To access the layout files from Java, user the “**R**” class
      5. R has a reference to all types of resources inside our **res** folder
      6. Since we are looking for our layout, we should access “**R.layout**” section
   4. Above code snippet is to set the content view, it is obvious that the view should be set upon the creation of the activity
   5. For this purpose, Activity superclass provides a method called “onCreate”, which will fire when an activity class is initiated
   6. The whole code snippet will look like below;  
        
        
      **(Note**:- You don’t have to write the code from scratch in Android Studio, it provides a comprehensive auto complete, and to use it just press “Ctrl + Space” anywhere in the editor)
5. Now our designed layout is bind to an activity class. Now run the app and check if you could see the new UI
6. **You will notice that still our old “HelloWorld” UI appears after running the application**
7. Let’s see how to change the activity which start up at the app startup in the next section

## How to change application's starting activity

As it was mentioned in earlier sections, such app configurations are stored in “**AndroidManifest.xml**” file.

1. Open the “**AndroidManifest.xml**” file and check its current content  
   
   1. The root tag of the file is <manifest> and notice the namespace url and the “**package**” attribute
   2. Inside the root tag the application is defines using **<application>** tags
   3. Note the attributes used in application tag such as “**theme**”, “**icon**”, “**label**”
   4. Inside the **<application>** tags, activities in the application are defined using **<activity>** tags
   5. You can see the “**HelloWorldActivity**” which was created at application creation inside the manifest file as shown above, note the below attributes of it;  
      <**action android:name="android.intent.action.MAIN"** />  
      <**category android:name="android.intent.category.LAUNCHER"** />
      1. To make an activity seen on the launcher you need to add these attributes to your activity in the manifest
   6. Since our new activity “**SendMessageActivity**” is not already defined in the “**AndroidManifest.xml**”, lets define it first using below code snippet;  
        
       **<activity android:name="lk.uok.mit.activity.SendMessageActivity"></activity>**
   7. **To make it the starting activity, cut the below code snippet from** “**HelloWorldActivity**” and paste it inside “**SendMessageActivity**” definition  
      **<intent-filter>  
       <action android:name="android.intent.action.MAIN" />  
       <category android:name="android.intent.category.LAUNCHER" />  
      </intent-filter>**
   8. Now the whole entry for “**SendMessageActivity**” will look like below;  
      
   9. The whole file will look like below;  
      
   10. **Save everything and run the app now and check the outcome;**

Now our new Ui has been displayed upon app startup

Note that, upon app startup, the “Contact Number” field is automatically focused and as a result, the keyboard appears in the screen

Click on the second text box “Message” and see how the keyboard layout changes from numbers to numeric.

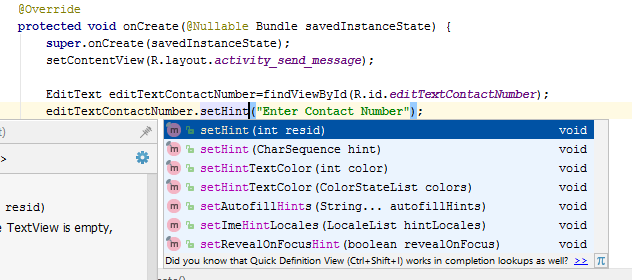
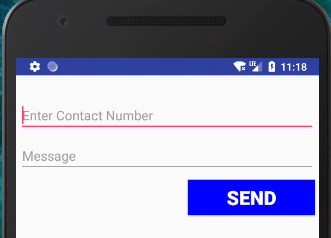
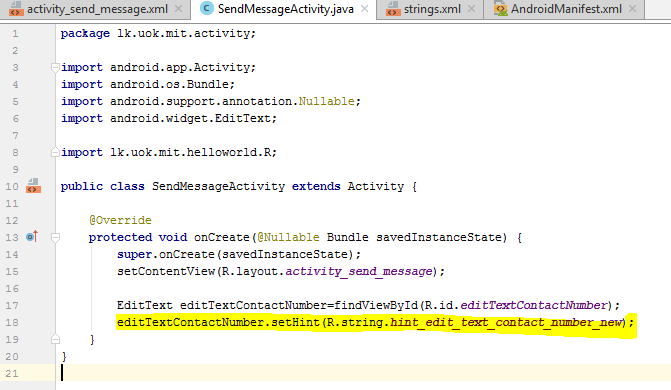
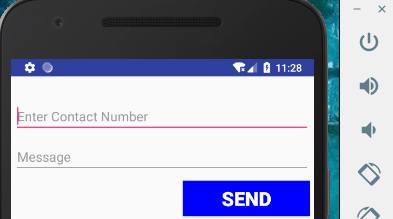
If you could recall, we made the type of “Contact number” as “Phone” and “Message” as “Text”

## Accessing views from the layout in an activity

In your activity (and fragment) code you frequently need to access the views to access and modify their properties.

* In an activity you can use the **findViewById(id)** method call to search for a view in the current layout.
* The id is the ID attribute of the view in the layout.
* The usage of this method is demonstrated by the following code.  
    
  **TextView textView = (TextView) findViewById(R.id.mytext);**
  + A text view has been defined in the layout with android:id=”mytext”, and it has been accessed in Java code like above

Now let’s see how we can change the label (android:hint) attribute of the “Contact Number” EditText which is having the android:id=”editTextContactNumber” attribute

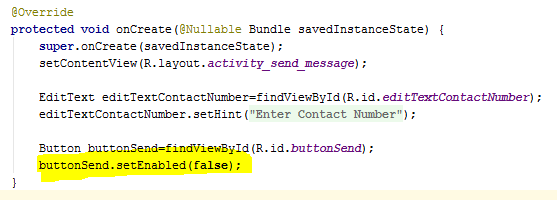
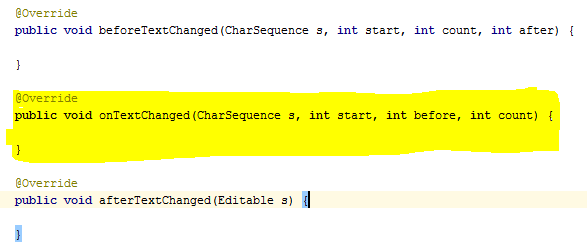
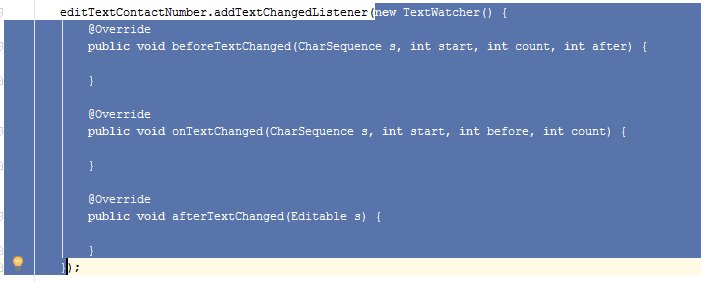
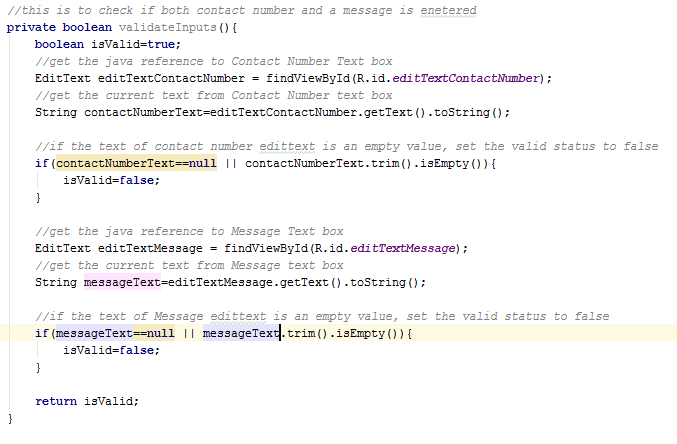
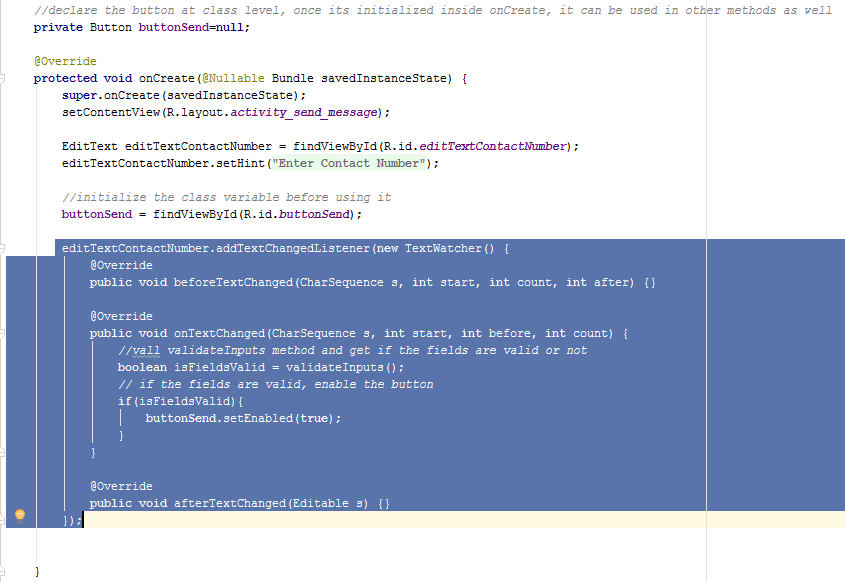
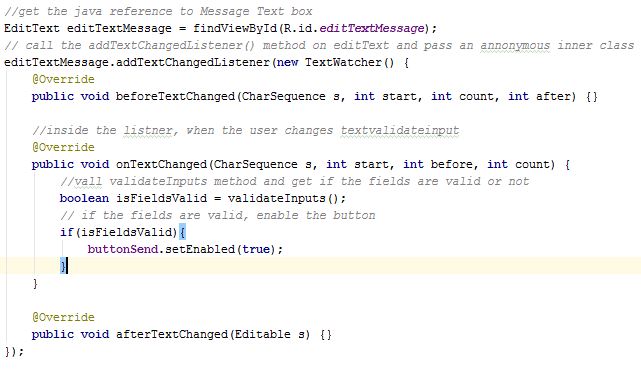
1. Open the “**SendMessageActivity**” class and inside “onCreate” method use below code to access the EditText with id “**editTextContactNumber**”  
     
   **EditText editTextContactNumber=findViewById(R.id.editTextContactNumber);**
   1. Now we have a reference to the layout’s EditText as **“editTextContactNumber”** from Java
2. Now change the “hint” attribute of the EditText as below;  
     
   **editTextContactNumber.setHint("Enter Contact Number");**
   1. There are setter methods in Java for every attribute in View we selected.
   2. You will note from the autocomplete, that there are several overloaded methods for “setView”, which will accept String, int etc.  
      
   3. Above we have used the method which accepts a “String” parameter, it is same as hardcoding the string value in XML
   4. Now save everything, and run the app and see the output, note the change in hint  
      
   5. Even inside the Java file, it is not recommended to hard code the String values, lets define a new string value in the “strings.xml” like below, **(we could change the value of the old entry easily, but this is for the demo purpose)**  
      **<string name="hint\_edit\_text\_contact\_number\_new">Enter Contact Number</string>**
   6. Now change the setHint method to use that string resource value in place of the hardcoded value like shown below;  
        
      **editTextContactNumber.setHint(R.string.hint\_edit\_text\_contact\_number\_new);**
      1. When we were accessing layouts using R class we used **R.layout**
      2. Here we want to access string resources and we are using **R.string**
   7. Now the whole code snippet will look like below;  
      
   8. Now run the app and check  
      

# Navigating between Activities

Single activity is not enough for an app. In a real world application, there can be multiple activities. We saw how to launch an activity at app startup, now lets see how can we navigate from our “**SendMessageActivity**” to the old “**HelloWorldActivity**”

Lets set our requirement as below,

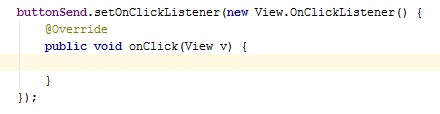
* When a user enters the contact number and a message and press “SEND” button, it should go to another UI and display the message
* The send button should be enabled only if a valid contact number and a message is entered

1. According to the requirement, the “**buttonSend**” should be in disabled state by default, there are two ways of doing this (use **android:enabled** attribute for this)
   1. Set it as **android:enabled** =”false”using “**activity\_send\_message.xml**” XML code  
      
   2. Get a reference to the **Button** from Java class and set the enabled status as false like blow  
      
   3. We cse either one of above methods to set the disabled status, let’s go for the XML version
2. Now we have disabled the “**buttonSend**”, now to activate the button, the user should enter both Contact Number and Message
   1. In order to do this we have to check both text boxes, and there is no guarantee if user will add details in the exact given order
   2. In such scenarios we can use **listeners**
      1. An event listener is an interface in the View class that contains a single callback method.
      2. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
   3. A listener can be fired each time the text value for a text box changes
   4. We have to add listeners for both text boxes, and inside each listener we should check if both text box have values
3. Add “**TextChangedListener**” for both EditText views
   1. The listener should be attached on UI creation – that is inside onCreate
   2. We already have a java reference to the “editTextContactNumber”,
   3. To bind a listener to detect any text change, there is a method called “**addTextChangedListener**” in EditText, the parameter it takes of type **TextWatcher**
      1. When an object of a type **TextWatcher** is attached to an Editable, its methods will be called when the text is changed.
      2. **TextWatcher** is an interface and any class it implements must implement below methods;  
         
         1. **afterTextChanged -** This method is called to notify you that, somewhere within **EditText s**, the text has been changed
         2. **beforeTextChanged -** This method is called to notify you that, within s, the count characters beginning at start are about to be replaced by new text with length after
         3. **onTextChanged -** This method is called to notify you that, within s, the count characters beginning at start have just replaced old text that had length before
      3. The highlighted method “**onTextChanged**” is the useful one for us, it gets fired when the relevant text boxes’ text changes
      4. There are two options here
         1. Create a class of Type “TextWatcher” (by implementing it), and get an instance to pass in to “**addTextChangedListener**”
         2. **Create an anonymous inner class**
      5. When we proceed with this tutorial series, it will be clear to you that most often, anonymous inner classes are used to add listeners in android
   4. below is the code which should be added to bind a listener to a text box  
      
      1. The highlighted part is the anonymous inner class of type “TextWatcher”
      2. Inside the “onTextChanged” method, we can call another method to check if both EditTexts are filled with valid data
   5. Below is a method to check if entered data in to both “editTextContactNumber” and “editTextMessage” are valid  
      
      1. Text values has been retrieved from both text boxes using there java references here
      2. That text value has been checked to see if they have entered at least one character in both text boxes
      3. Please see the comments to understand the functionality of each line
   6. Now let’s call this method when the user enters text in either of text boxes, and when this method return true, it means we can enable the “Send” button
   7. Within “onTextChangedMethod”, call the “validateInputs” method like shown below;  
      
      1. Note the highlighted area, and refer comments to understand code
   8. In the same way, implement the “**addTextChangedListener**” to “**editTextMessage**” as shown below, this snippet should also come inside “onCreate” method of the activity  
      

There were two conditions in the requirement like below, and the second condition is done after implementing the code given above

* When a user enters the contact number and a message and press “SEND” button, it should go to another UI and display the message
* The send button should be enabled only if a valid contact number and a message is entered – Done

Now we should consider the “SEND” button’s click action, what should happen when user clicks “SEND” button.

1. As same as above, first we have to get the user’s action on click.
   1. For this purpose, there is a method similar to “**addTextChangedListener**” as “**setOnClickListener**”
   2. In to “**setOnClickListener**”, we should pass a parameter of type “**OnClickListener**”
   3. Same as “TextWatcher” above this too is an interface, however unlike the “**TextWatcher**”, “**OnClickListener**” is defined inside “**View**” class, and should access accordingly
      1. E.g:- **View.OnClickListener**
   4. Therefore we can pass an anonymous inner class in to this methods as the parameter
   5. The code snippet will look like below,  
      
      1. There’s only one method as “onClick” here
      2. Inside “onClick”, we should write a method to fulfill below two requirements given;
         1. should go to another UI
         2. display the message (in second UI)
2. Before implementing the rest of requirement, let’s check if the code written so far works or not
   1. We can fire an event inside onClick method and check if button is working
   2. Above will validate all our work so far, since a disabled button cannot be clicked
   3. To test if the click event of the “SEND” button works, we can add a Toast

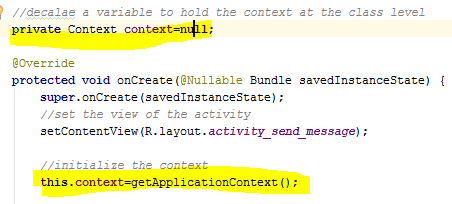
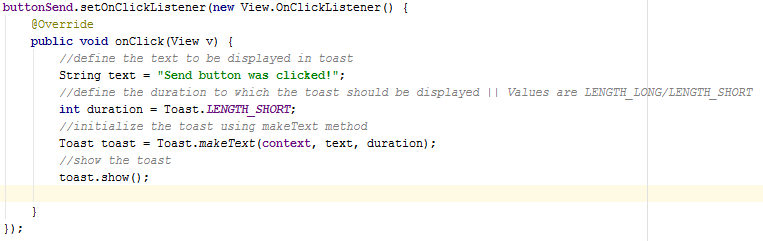
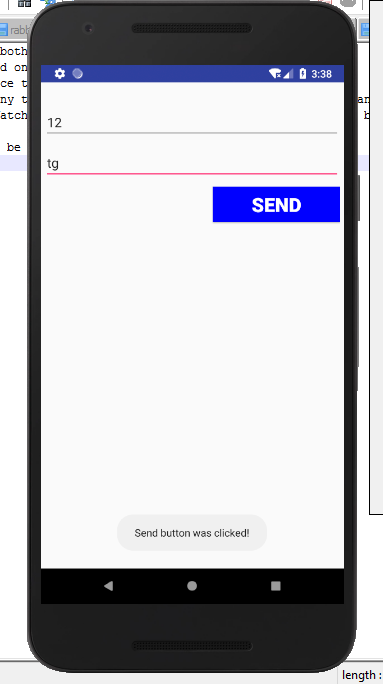
# Toasts

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

First, we need to instantiate a Toast object with one of the makeText() methods. This method takes three parameters:

* the application Context,
* the text message, and
* the duration for the toast.

It returns a properly initialized Toast object. You can display the toast notification with show() method.

1. Toast requires an application context, but we are trying to display toast from inside an anonymous inner class.
   1. Therefore we need a reference to the application context.
   2. For this declare a variable of type **android.content.Context** at the class level and initiate it inside “onCreate” method  
      
      1. As a habit do this in every activity class, inside onCreate, after the setContentView method
2. Now using below code snippet we can display a toast when user clicks on the button  
   
3. To check if everything done so far is working, run the application and check by filling contact number. Message and clicking on “SEND” button  
   

Enter values to both text boxes and click on “SEND”

The toast will appear at the bottom of the screen as shown here

1. Now we have confirmed that the click event on button is working. As the next step, lets see how to navigate to the “HelloWorldActivity” from our current “SendMessageActivity”

# Intents

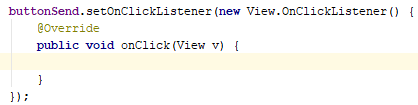
An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways,   
**there are three fundamental use cases**:

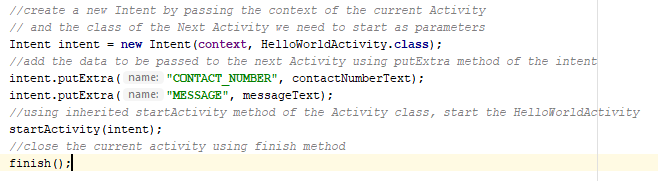
* **Starting an activity**
  + An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity().
  + The Intent describes the activity to start and carries any necessary data.
  + If you want to receive a result from the activity when it finishes, call startActivityForResult().
  + Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback.
* **Starting a service**
  + A Service is a component that performs operations in the background without a user interface.
  + With Android 5.0 (API level 21) and later, you can start a service with JobScheduler
  + For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the Service class.
  + You can start a service to perform a one-time operation (such as downloading a file) by passing an Intent to startService().
  + The Intent describes the service to start and carries any necessary data.
  + f the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService().
* **Delivering a broadcast**
  + A broadcast is a message that any app can receive.
  + The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging.
  + You can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast()

From above three scenarios, we are considering the usage of intent in “**Starting an activity**” here.

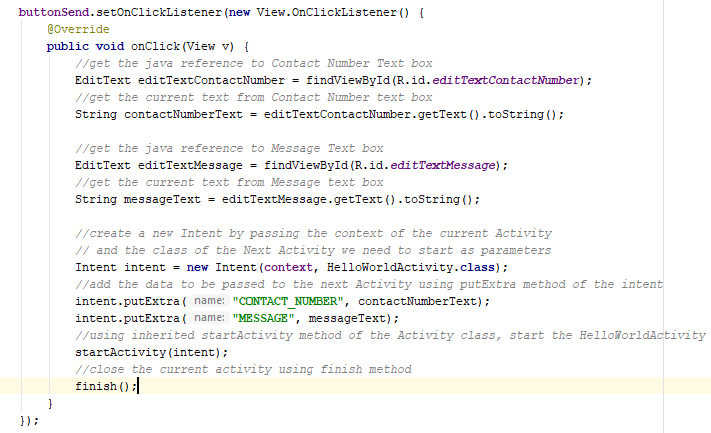
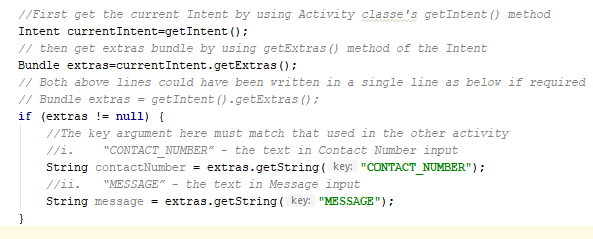
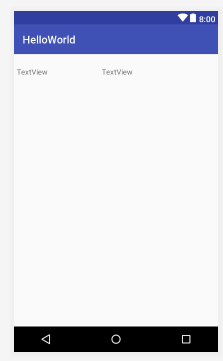
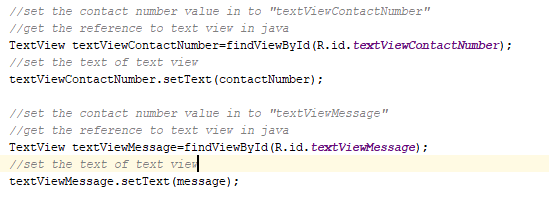
Below are the steps to write code to navigate to “HelloWorldActivity” from our current “SendMessageActivity”. That is, we need to

* **start** “**HelloWorldActivity**”
* **close** the “**SendMessageActivity**”

1. Open “**SendMessageActivity”,** and remove the code we added to display the Toast from “click” method of “SEND” button, it should look like below  
     
   
2. Below is the code to start a new activity using an Intent,
   1. if you could recall. We had a requirement to display the message in the second Activity,
   2. for that we should pass the message text to the next activity



* 1. The first line is to **create** an **Intent,** as the parameters, **the context of the current activity** and the **class of the next Activity** class we need to start should be passed
  2. To pass data in to the next activity, we should enter them using “**putExtra**” method of the Intent
     1. These are added as key-value pairs
  3. To start a new activity, there is an inherited method from Activity class as “**startActivity**”, to that we should pass out new Intent instance
  4. Finally, to finish the current activity, call finish() method of Activity class

1. After adding above to the “click” event, the whole code snippet will look like below,  
   
2. Next step is to display the message we received from the “**SendMessageActivity”** in the **“HelloWorldActivity”**
   1. Open **“HelloWorldActivity”** class, and the current code will look like below  
        
      
   2. First, we need to receive the data passed from previous activity, data was passed using two keys as;
      1. “CONTACT\_NUMBER” – the text in Contact Number input
      2. “MESSAGE” – the text in Message input
   3. Below is the code that should be used to receive those values inside “HelloWorldActivity”  
      
      1. First get the current Intent by using Activity classe's **getIntent()** method
      2. then get extras bundle by using **getExtras()** method of the Intent
      3. Since both values we added were “**Strings**” we can call “**getString**” method on bundle instance to retrieve our values
   4. Now we have the values from previous activity in our “HelloWorldActivity” class, now lets modify the “**activity\_hello\_world.xml**” by adding two text views as shown below  
      
      1. Add two TextViews with following ID s
         1. **textViewContactNumber – to display contact number**
         2. **“textViewMessage” – to display message**
      2. The design view should look like below;  
         
   5. Having the two text views added, add below code to display the passed values in the new Activity  
      
      1. Same as we did before, references to both textViews in java and set the text as given above
3. Now we can run the program and check if everything works

# 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