|  |
| --- |
| Android Tutorial – Part 3 |

|  |
| --- |
| 6-11-2018 |



Table of Contents

[Introduction 2](#_Toc516607171)

[Load contact List in the phone to a search box 3](#_Toc516607172)

[AutoCompleteTextView 4](#_Toc516607173)

[Class hierarchy of “**AutoCompleteTextView**” 4](#_Toc516607174)

[Important methods of AutoCompleteTextView 4](#_Toc516607175)

[Adapters in Android 5](#_Toc516607176)

[Why Adapters 6](#_Toc516607177)

[Commonly used Adapters in Android 6](#_Toc516607178)

[Cursors 6](#_Toc516607179)

[References 7](#_Toc516607180)

# Introduction

This is the part three 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 tutorial, opened in Android Studio.
* AVD or an Actual device ready for app deployment.

To catch up, in the last session (Android Tutorial Part 2),

* Android Studio’s UI Design View (Layout Editor)
* Android studio functionality
  + How to Rename a File
  + How to Compare Two Files
  + How to Delete a File
* Android Views and View Groups
  + Different Types of Views
* Design a User interface
  + 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
* Access and Modify UI from Activity class
  + **setContentView** Method
  + change application's startup activity
  + **findViewById** Method
  + **TextChangedListener**
  + **OnClickListener**
* Toasts
* Intents
  + Navigating between Activities
* Source code for this part can be found in Git Repository given below :- <https://github.com/nadee158/android_tutorial_part_2.git>

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

* Load contact List in the phone to a search box
  + Adding special permissions to “**AndroidManifest.xml**”
* Using a SQLite Database in Android Application

# Load contact List in the phone to a search box

First, recall the “**SendMessageActivity**” and “**activity\_send\_message.xml**”, from the last tutorial. The requirement which was given, and the resulted design was as below;

|  |  |
| --- | --- |
| Requirement:- | Result:- |
|  |  |

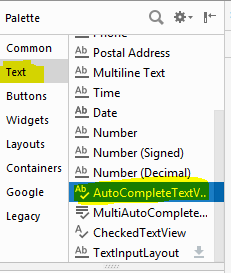
In the above requirement, it was only asked to let the user type the desired contact number in to the text box. Instead of doing this, it will be a lot easier for the user, if he could simply select from the existing contact list in the phone.

Therefore, as an enhancement to the work we already did, let’s make the “Contact Number” filed, a search from contact list rather than a text box.

Modified Requirement:-

* ***Let the user select Contact Number from the contact list rather than asking to type in***
* ***If the Contact Number is not in the list, allow the user to type the number***

In order to match the above requirement, we need to choose an appropriate view to replace the existing “**EditText**” component with id “**editTextContactNumber**”. For this purpose, a view exists in Android as “**AutoCompleteTextView**”, which is available under “**Text**” category in “**Palette**” view in android studio.



## AutoCompleteTextView

This is an editable text view that shows completion suggestions automatically while the user is typing in android apps, so no need to write all the characters of the word.

* Is an editable text field.
* Displays a list of suggestions in a drop down menu from which user can select only one suggestion or value.
* The list of suggestions is obtained from an **adapter** and it appears only after a number of characters that are specified in the threshold.
* Is a subclass of “**EditText**” class.
* The “**MultiAutoCompleteTextView**” is the subclass of “**AutoCompleteTextView**” class.

### Class hierarchy of “**AutoCompleteTextView**”

Object

View

TextView

EditText

AutoCompleteTextView

MultiAutoCompleteTextView

.

### Important methods of AutoCompleteTextView

* **getAdapter**() : This method returns a filterable **list adapter** used for auto completion
* **getCompletionHint**() : This method returns optional hint text displayed at the bottom of the matching list
* **getDropDownAnchor**() : This method returns the id for the view that the auto-complete drop down list is attached to
* **getListSelection**() : This method returns the position of the dropdown view selection, if there is any
* **isPopupShowing**() : This method indicates whether the popup menu is showing
* **setText**(CharSequence text, boolean filter) : This method sets text except that it can disable filtering
* **showDropDown**() : This method displays the drop down on screen
* **setAdapter** method is used to set the adapter of the “**AutoCompleteTextView”**.

## Adapters in Android

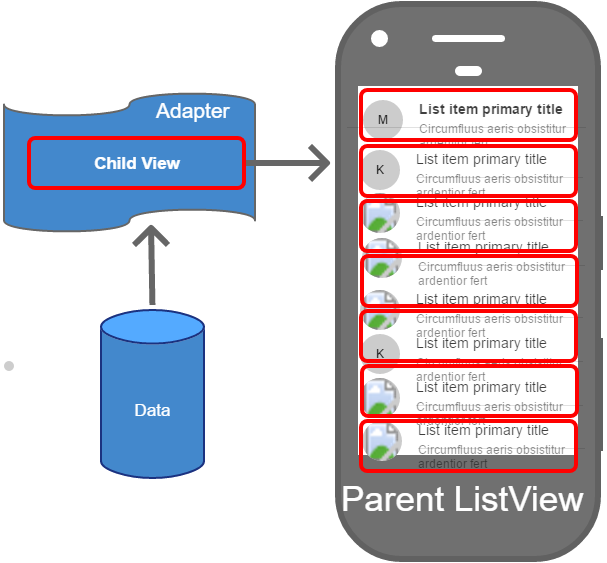
If you could notice, the term “Adapter” came up when referring to a “**AutoCompleteTextView”** above. Before going in to coding and design part with “**AutoCompleteTextView”**, it is important to understand the “**Adapter**” s available in Android and their purpose.

In technical terms;

* An adapter is an object of a class that implements the Adapter interface.
  + It acts as a link between a data set and an adapter view,
* It is an object of a class that extends the abstract “**AdapterView**” class.
* The data set can be anything that presents data in a structured manner.
  + Arrays, List objects, and Cursor objects are commonly used data sets.

In simple Terms;

* **Adapter** is a **bridge** between **UI component** and **Data Source.**
  + Helps to fill data in UI component.
* Holds the data and send the data to an Adapter view.
  + then view takes data from the adapter view and shows on different views like as ListView, GridView, Spinner, “**AutoCompleteTextView”,** etc.
* For more customization in Views base adapter or custom adapters can be used.



UI Component, can be a **ListView**, **Spinner**, **AutoCompleteTextView**, etc. View Item s are displayed here

Adapter; Acts as a bridge, and convert **data item** to **view item**

Data source; can be ArrayList, HashMap, SQLite DB, etc.

### Why Adapters

Adapter views can display large data sets very efficiently. For instance, the **ListView** and **GridView** widgets can display millions of items without any noticeable lag while keeping memory and CPU usage very low. Different adapter views follow different strategies. Most of them are as below,

* They render only those View objects that are either already on-screen or that are about to move on-screen.
  + This way, the memory consumed by an adapter view can be constant and independent of the size of the data set.
* They also allow developers to minimize expensive layout inflate operations and recycle existing View objects that have move off-screen.
  + This keeps CPU consumption low.

### Commonly used Adapters in Android

* **BaseAdapter** – It is parent adapter for all other adapters
* **ArrayAdapter** – It is used whenever we have a list of single items which is backed by an array
* **SimpleCursorAdapter** - The SimpleCursorAdapter links the data contained in a **Cursor** to an Adapter View
  + Binds the Cursor data to an Adapter View.
  + Can define a layout that controls how each row of data is displayed.
  + Each row’s view is populated using the column values of the corresponding row in the cursor.
  + This layout is then displayed in the Adapter View, like a **ListView** for example.
* **CursorAdapter** - A CursorAdapter links a Cursor’s data to a List View.
  + Must include the database’s \_id column as it’s used in processing the list item’s selection.
  + The SimpleCursorAdapter is a subclass of CursorAdapter.
  + The SimpleCursorAdapter is easier to use while the CursorAdapter requires more work but allows more customization.
* **SimpleAdapter** – It is an easy adapter to map static data to views defined in the XML file
* **ListAdapter** - links the data and a **ListView** displaying the data.
  + The List View can display any data type provided it’s wrapped in a ListAdapter.

### Cursors

A cursor is a set of data. Usually a cursor can be obtained when doing a database query. The result of the query is contained in the cursor.

* The basic purpose of a cursor is to point to a single row of the result fetched by the query.
* Load the row pointed by the cursor object.
* By using cursor, can save lot of ram and memory.

In the next section of this tutorial, further use of “Cursor” will be demonstrated

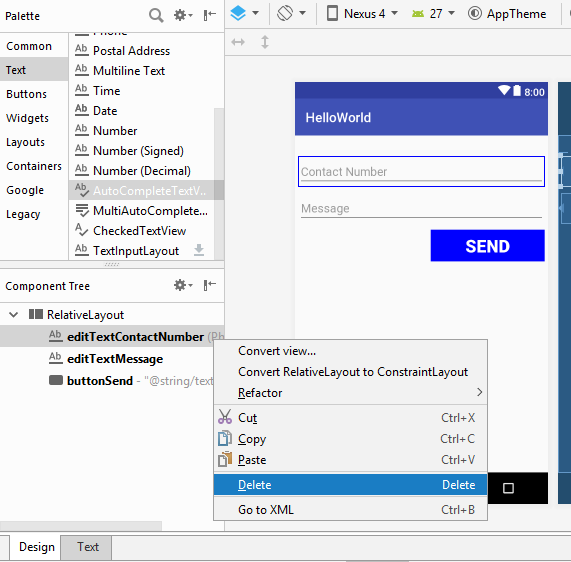
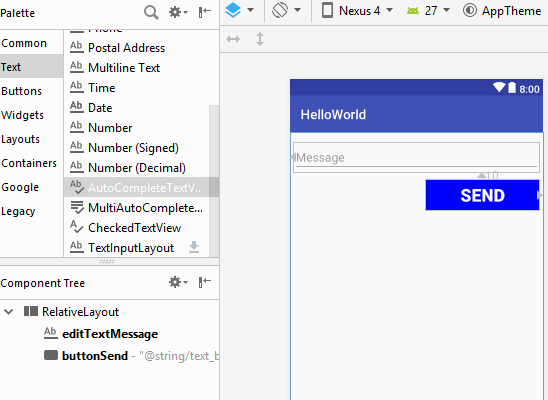
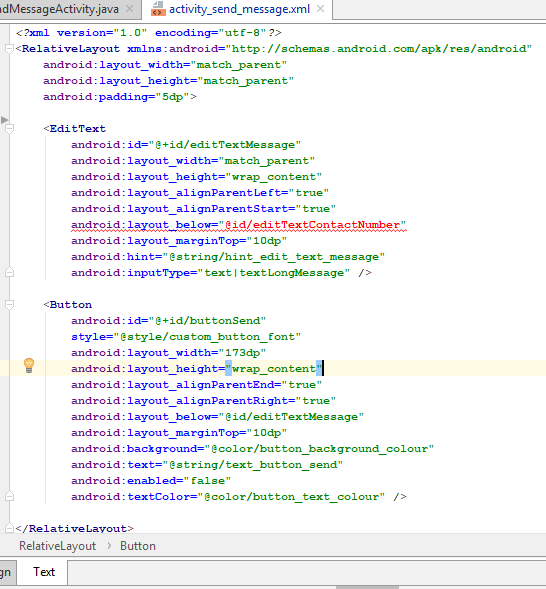
With the current theoretical understand of an “**AutoCompleteTextView**” and “**Adapter**”, lets load the contact list of the device in to a **AutoCompleteTextView** in our “**SendMessageActivity**” and “**activity\_send\_message.xml**”.

## User Interface Design – Modification of **activity\_send\_message.xml**

Regarding the user interface (layout.xml) file, one change is required to the current design. That is to replace the “**EditText**” view with id “**editTextContactNumber**” from “**activity\_send\_message.xml**” layout file and add an “**AutoCompleteTextView**” view in its position with id “**autoCompleteTextContactNumber**”.

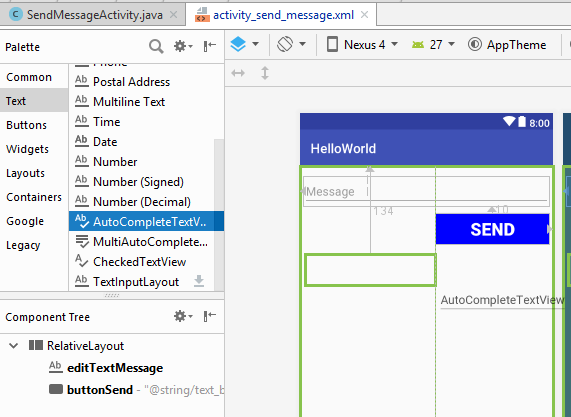
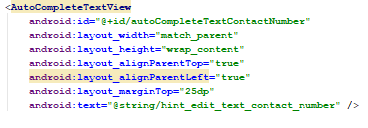
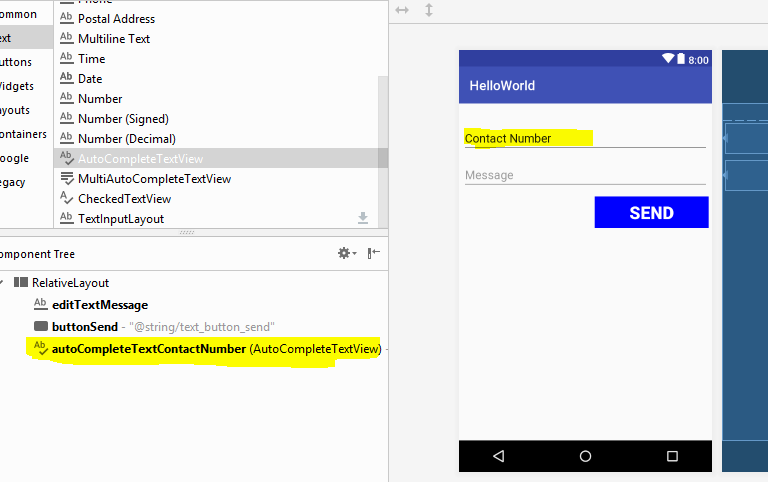
### Remove the EditText from Layout

To remove the “**EditText**” view with id “**editTextContactNumber**” from “**activity\_send\_message.xml**” layout;

1. Open “**activity\_send\_message.xml**” in res/layout and go to “Design” view
2. On “**Component Tree**” view, right click on “**editTextContactNumber**”, and click “**Delete**”
3. After deleting, the layout will look like below;  
   
4. Go to “Text” view of “**activity\_send\_message.xml**”, it will look like below;  
   
   1. Notice the error in layout, it is because the id of the deleted “EditText” is still used. (Ignore it for now)

### Add a AutoCompleteTextView to Layout

To add an “**AutoCompleteTextView**” view with id “autoCompleteTextContactNumber” to “**activity\_send\_message.xml**”, and position it in the same place as deleted “EditText” view’s;

1. Open “**activity\_send\_message.xml**” in res/layout and go to “Design” view
2. Drag a “**AutoCompleteTextView**” under the category “**Text**” from “**Palette**” and drop in to any place on “**activity\_send\_message.xml**” layout file  
   
   1. Lets do the positioning by editing XML code this time.
3. To position the dragged “**AutoCompleteTextView**”, go to the “Text” view of “**activity\_send\_message.xml**”, it should look like below (apart for the positioning attributes)  
   
4. Change below attributes in the XML file to **position** the UI component properly
   1. **android:layout\_alignParentTop="true"** - aligns the view to the top of parent layout (view group)
   2. **android:layout\_alignParentLeft="true"**- aligns the view to the left of parent layout (view group)
   3. **android:layout\_marginTop="25dp"** – keeps a margin of 25 dp (density) units from top of the parent layout
5. Change below attributes in the XML file to **resize** the UI component properly
   1. **android:layout\_width="match\_parent"** – makes the width as same as the parent layout
   2. **android:layout\_height="wrap\_content"** – makes the height based on the content of the view
6. Give the requested id to the view
   1. **android:id="@+id/autoCompleteTextContactNumber"** – gives the id **“autoCompleteTextContactNumber”** to the “**AutoCompleteTextView**”
7. Change the reference to **“editTextContactNumber”** in **“editTextMessage”** like below;
   1. **android:layout\_below="@id/autoCompleteTextContactNumber"** – position the “EditText” view for “Message” with id “**editTextMessage**” under the new view
   2. **Note:-** the above property should be changed in “**editTextMessage**” EditText, in order to fix the error mentioned above
8. Change the text of the view
   1. **android:text="@string/hint\_edit\_text\_contact\_number"** – give the created string resource in previous tutorial as the text of the new view
9. Now the XML code of “**AutoCompleteTextView**” in “**activity\_send\_message.xml**” should look like below;  
   
10. The design view of “**activity\_send\_message.xml**” should look like below;  
    
11. With this, the UI design part is completed. Next,
    1. Any references to “**editTextContactNumber**” in Java code should be removed (Already there will are compile errors in Activity class)
    2. The contact list from phone should be loaded
    3. the adapter should be created,
    4. adapter should be bind in to the “**autoCompleteTextContactNumber**”

# References

*Android AutoCompleteTextView Example*. (2018, January 1). Retrieved from JavaTPoint: https://www.javatpoint.com/android-autocompletetextview-example

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

CHUGH, A. (2018, April 2). *Android AutoCompleteTextView Example Tutorial*. Retrieved from JournalDev: https://www.journaldev.com/9574/android-autocompletetextview-example-tutorial

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