**Элементы пользовательского интерфейса**  
Спиннер, ресурс-массив, адаптеры

## Typed array

A [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray?hl=ru) defined in XML. You can use this to create an array of other resources, such as drawables. Note that the array is not required to be homogeneous, so you can create an array of mixed resource types, but you must be aware of what and where the data types are in the array so that you can properly obtain each item with the [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray?hl=ru)'s get...() methods.

**Note:** A typed array is a simple resource that is referenced using the value provided in the **name** attribute (not the name of the XML file). As such, you can combine typed array resources with other simple resources in the one XML file, under one **<resources>** element.

**file location:**

res/values/filename.xml  
The filename is arbitrary. The <array> element's name will be used as the resource ID.

**compiled resource datatype:**

Resource pointer to a [TypedArray](https://developer.android.com/reference/android/content/res/TypedArray?hl=ru).

**resource reference:**

In Java: R.array.array\_name  
In XML: @[package:]array.array\_name

**syntax:**

<?xml version="1.0" encoding="utf-8"?>  
<[**resources**](https://developer.android.com/guide/topics/resources/more-resources?hl=ru#array-resources-element)>  
    <[**array**](https://developer.android.com/guide/topics/resources/more-resources?hl=ru#array-element)  
        name="*integer\_array\_name*">  
        <[**item**](https://developer.android.com/guide/topics/resources/more-resources?hl=ru#array-item-element)>*resource*</item>  
    </array>  
</resources>

**elements:**

<resources>

**Required.** This must be the root node.

No attributes.

<array>

Defines an array. Contains one or more child <item> elements.

attributes:

android:name

String. A name for the array. This name will be used as the resource ID to reference the array.

<item>

A generic resource. The value can be a reference to a resource or a simple data type. Must be a child of an <array> element.

No attributes.

**example:**

XML file saved at res/values/arrays.xml:

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <array name="icons">  
        <item>@drawable/home</item>  
        <item>@drawable/settings</item>  
        <item>@drawable/logout</item>  
    </array>  
    <array name="colors">  
        <item>#FFFF0000</item>  
        <item>#FF00FF00</item>  
        <item>#FF0000FF</item>  
    </array>  
</resources>

This application code retrieves each array and then obtains the first entry in each array:

[KOTLIN](https://developer.android.com/guide/topics/resources/more-resources?hl=ru#kotlin)[JAVA](https://developer.android.com/guide/topics/resources/more-resources?hl=ru#java)

val icons: TypedArray = [**resources**](https://developer.android.com/reference/android/content/Context?hl=ru#getResources()).[**obtainTypedArray**](https://developer.android.com/reference/android/content/res/Resources?hl=ru#obtainTypedArray(int))(R.array.icons)  
val drawable: Drawable = icons.[**getDrawable**](https://developer.android.com/reference/android/content/res/TypedArray?hl=ru#getDrawable(int))(0)  
  
val colors: TypedArray = [**resources**](https://developer.android.com/reference/android/content/Context?hl=ru#getResources()).[**obtainTypedArray**](https://developer.android.com/reference/android/content/res/Resources?hl=ru#obtainTypedArray(int))(R.array.colors)  
val color: Int = colors.[**getColor**](https://developer.android.com/reference/android/content/res/TypedArray?hl=ru#getColor(int,%20int))(0,0)

**Table of Contents**[[hide](https://www.journaldev.com/394/android-listview-using-kotlin)]

* [1 Android ListView](https://www.journaldev.com/394/android-listview-using-kotlin#android-listview)
  + [1.1 Setting Selectors and Divider Color](https://www.journaldev.com/394/android-listview-using-kotlin#setting-selectors-and-divider-color)
  + [1.2 Adapter](https://www.journaldev.com/394/android-listview-using-kotlin#adapter)
  + [1.3 Types of Adapters](https://www.journaldev.com/394/android-listview-using-kotlin#types-of-adapters)
  + [1.4 Project Structure](https://www.journaldev.com/394/android-listview-using-kotlin#project-structure)
  + [1.5 Code](https://www.journaldev.com/394/android-listview-using-kotlin#code)

**Android ListView**

ListView is a very common UI element in Android Applications. It is used to display a list of items separated by dividers which can be scrolled endlessly. It’s generally used to display a group of related items.

Adding a ListView to your layout:

<ListView

android:id="@+id/recipe\_list\_view"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content />

Following are the key XML attributes used inside the ListView:

* android:divider : Drawable or color to draw between list items.
* android:divider : Drawable or color to draw between list items.
* android:dividerHeight : Height of the divider.
* android:entries : An array resource can be passed here to be displayed as a List.
* android:footerDividersEnabled : When set to false, there will be no divider at the end of the ListView.
* android:headerDividersEnabled : When set to false, there will be no divider at the top of the ListView
* android:clickable: This makes the ListView rows clickable. If you’re setting the ListView from the Activity, setting this attribute is not neccessary
* android:listSelector: Set the background color of the list view row when it is clicked.

We can populate entries in the ListView without any Java code by defining an array in the strings.xml file:

<string-array name="Colors">

<item name="color">Red</item>

<item name="color">Orange</item>

<item name="color">Yellow</item>

<item name="color">Green</item>

<item name="color">Blue</item>

<item name="color">White</item>

<item name="color">Black</item>

<item name="color">Purple</item>

<item name="color">Pink</item>

<item name="color">Gray</item>

<item name="color">Cyan Blue</item>

<item name="color">Magenta</item>

</string-array>

Now the ListView is populated in the layout as:

<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<ListView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:entries="@array/Colors"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

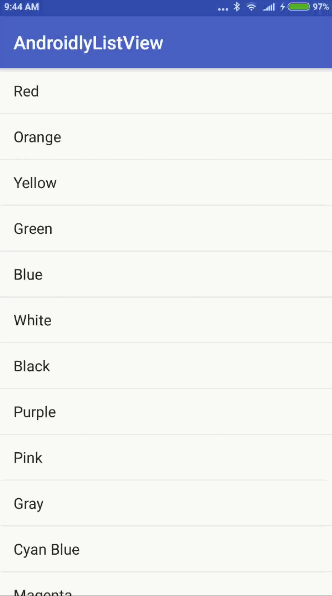
app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

Setting the width to wrap\_content wraps the ListView rows to its content.

Following is how it looks on the screen:



**Setting Selectors and Divider Color**

Use the following ListView tag:

<ListView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:divider="@color/colorPrimary"

android:dividerHeight="1dp"

android:entries="@array/Colors"

android:listSelector="@android:color/holo\_green\_dark"

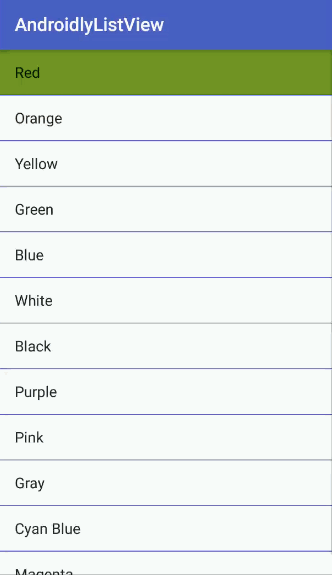
android:clickable="true"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />



The above code makes the ListView selectable. But to add a logic on each ListView row click we need to use the Adapter.

**Adapter**

A ListView class by itself cannot populate the entries. An Adapter is responsible for populating the data in the ListView. We have built-in adapter classes(like the one above) which come with a built-in layout for each row. We can create our own Custom Adapter classes as well.

The adapter comes with its own set of built-in methods. Following two are the most important ones:

getView() : We can inflate our own layouts in the Adapter inside this method.  
notifyDataSetChanged() method on the adapter is called if the data has changed or if new data is available.

To set the adapter on the ListView, we use the method setAdapter().

**Types of Adapters**

There are four main types of Adapters:

**BaseAdapter** – As the name suggests this abstract is extended by all the other adapters. When creating a custom adapter using this as the parent class, you need to override all the methods mentioned above along with getCount(), getId() etc.

**ArrayAdapter** – This populates the ListView with the array supplied. It is defined as:

var arrayAdapter = ArrayAdapter<String>(context,layout,array);

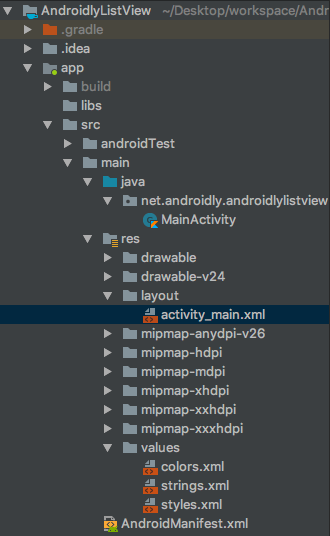
The first parameter is the context, followed by the layout resource for the list rows.  
The layout must have a TextView. The third parameter is the array.

For the ArrayAdapter you need to override the getView() method only. getCount() isn’t necessary since ArrayAdapter calculates the size of the array on its own.

**ListAdapter** – Unlike an ArrayAdapter, this is an interface. So it can be used only with concrete adapter classes. Concrete Adapter classes are ListActivity and ListFragment.

**SimpleCursorAdapter** – This is used when the data needs to populated from a Database. In its constructor, we must specify the layout for each row and also the Cursor instance which contains the fields that need to be displayed. We’ll look into this when we cover Databases.

**Project Structure**



**Code**

The code for the activity\_main.xml layout is given below:

<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<ListView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:entries="@array/Colors"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

The code for the MainActivity.kt class is given below:

package net.androidly.androidlylistview

import android.support.v7.app.AppCompatActivity

import android.os.Bundle

import android.widget.ArrayAdapter

import kotlinx.android.synthetic.main.activity\_main.\*

import org.jetbrains.anko.toast

class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

var colorArrays = resources.getStringArray(R.array.Colors)

var arrayAdapter = ArrayAdapter(this, android.R.layout.simple\_list\_item\_1, colorArrays)

listView.adapter = arrayAdapter

listView.setOnItemClickListener { adapterView, view, position: Int, id: Long ->

toast(colorArrays[position])

}

}

}

In the above code, we display a toast message. We arrived at that shorthand Toast function by adding the [Anko commons](https://www.androidly.net/307/android-anko-commons" \t "_blank) dependency in our build.gradle.

resources.getStringArray(R.array.Colors) converts the string array stored in the resources file into a Kotlin Array.

android.R.layout.simple\_list\_item\_1 is a built-in layout which hosts a TextView only

setOnItemClickListener is the Kotlin function that gets triggered when any ListView row is clicked. Here we can implement our logic. The four arguments inside the function are:

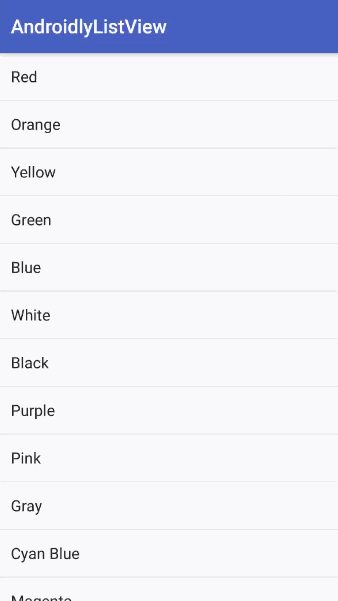
* adapterView: The parent view where the selection happens. Its ListView here.
* view: The selected view (row) within the ListView
* position: The position of the row in the adapter. This is an Int.
* id: The row id of the selected item. This is a Long.

Instead of retrieving the value using the array, we can get it from the adapterView as:

val selectedString = adapterView.getItemAtPosition(position) as String

getItemAtPosition returns the List View Row at that index. The row is a TextView in this case.

The output of the application in action is given below:



We can change the default item press color by creating a selector drawable inside the drawable folder.

list\_selector.xml

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

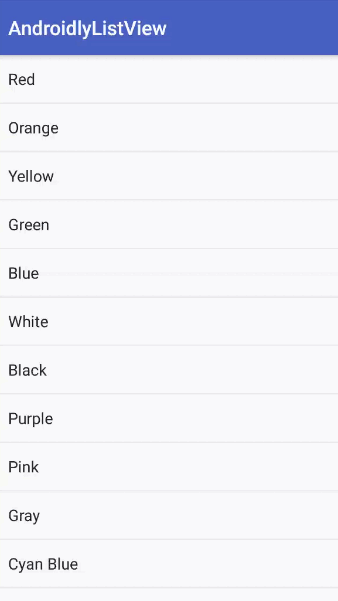
<item android:drawable="@color/colorAccent" android:state\_pressed="true"/>

<item android:drawable="@android:color/transparent"/>

</selector>

Add the following attribute inside the ListView XML Tag:

android:listSelector="@drawable/list\_selector"



This brings an end to this tutorial. You can download the project from the link below: