IN721 2019 Practical 1.2b – More on Resources

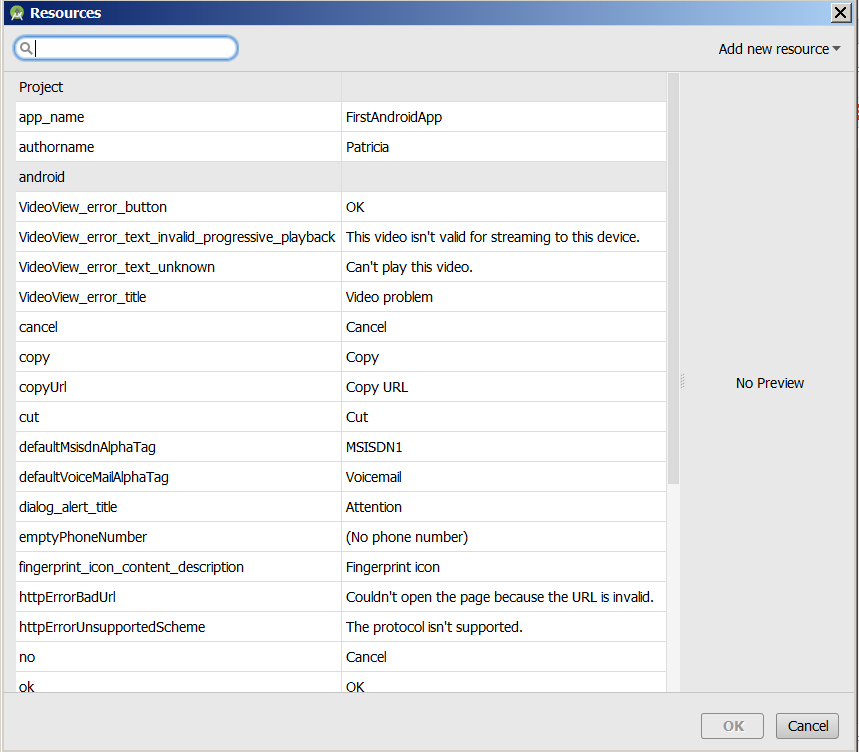
As discussed in lecture, Android does not want you to use literal values -- not for spatial dimensions, not for colours, not for strings. Instead, you are to define all these entities as XML elements in files in the res/values/ directory. A new Android Studio project comes with auto-generated files colors.xml, strings.xml, dimens.xml and styles.xml. You can add others as needed (see below).

## Accessing Resources - Three Ways

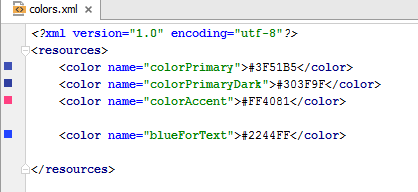
*Please read this material carefully before attempting the task at the end of the handout. Some is a review of material covered in class lecture, but some of it is new...*

Resource constants can be accessed from the graphical screen designer, from the XML of layouts, and from the Java code-behind of Activity classes

**Accessing resources from the Graphical Layout properties window:** You can type resource constants directly into the Properties pane, prefacing them with @*resourcetype/*, just as you do in the XML source (cf. the lecture)*.* Alternatively, clicking the ellipsis at the right-hand edge of any property entry text area will bring up a list of available resource constants and, for colours, a colour picker. Note that these lists will contain not only the resource constants you have defined, but also all the available system resource constants. For example, clicking the ellipsis for the Text property of a TextView brings up the window shown below. Using the Android system constants where appropriate will give your Android apps a more standardised interface. After you complete this practical, you may want to explore some of these system resource constants.



**Accessing resources in XML layout files:** (Also discussed in lecture.)Assume that you have added the constant blueForText to the default entries in /res/values/colors.xml, like this:



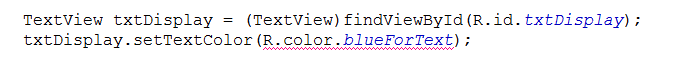
You can set the text colour of a TextView control to this colour in the layout XML file with:



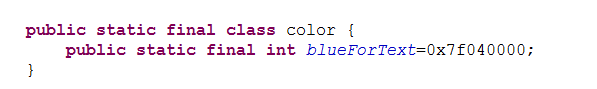
Remember that “@color/blueForText” is not a file path, it is a reference to an XML element of type <color> with name attribute blueForText.

**Accessing resources in the Java code-behind:**

Based on the code you have written so far, you might expect to be able to use resource constants in your Java code by asking R for them, like this:



But you can't – this is illegal. (Note the compiler complaining with its wiggly red underline.) The problem is that setTextColor wants a colour value. (Similarly setText wants a string, setWidth wants a pixel dimension, etc.) But R.color.blueForText is the ***resource id*** of the blueForText colour constant, not the actual colour value. You can see this in the class R, where R.color.blueForText is defined as an int.



When we use findViewById to get a reference to a screen control, this is not a problem, because findViewByID ***wants the resource id*** of the control it is fetching for you. But the various set methods want the ***values*** of constants, not their resource IDs. To deal with this, we need to get access to an instance of an Android helper class (specifically class Resources). Instances of this class know how to accept ***resource IDs*** of constants and return the associated ***values***.

The process involves three steps:

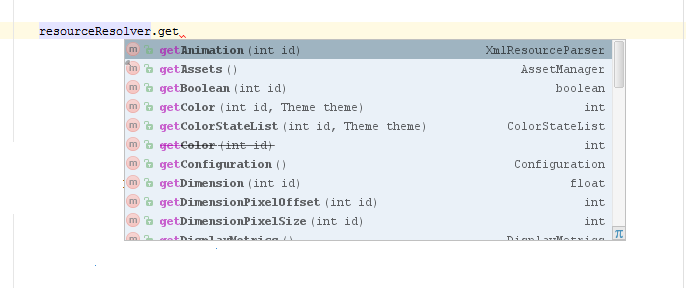
1. Get an object of type Resources. This object has methods to get resource constant values based on their Resource IDs.
2. Call the appropriate method of your Resources object, passing in an appropriate resource ID, and store the returned value.
3. Use the stored result.

We look at the code for each step in turn (please read all of this before starting the next task):

1. *Get an object of type Resources:* All members of the Activity class family expose a method called getResources(), which returns an instance of the Resources class. Look carefully at this pattern, because we will see it often. Android development relies heavily on the use of helper classes, and Activities have methods that generate instances of these helpers. The code in this case looks like:



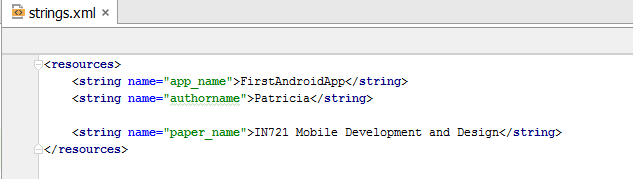
1. *Call the appropriate method of your Resources object, passing in an appropriate ID:* Objects of the Resources class expose many methods which accept a Resource ID, and return the value of the associated constant. Fortunately, you don’t have to memorise all of these methods, because Android Studio’s excellent code completion will help you, as illustrated:

**

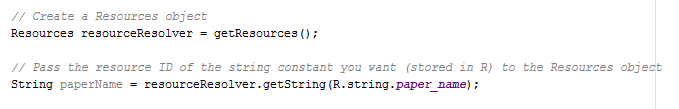
**Warning:** Note that the getColor(int id) method has been struck through. This indicates a method that belonged to an older version of the API, and has since been deprecated. Ongoing support for deprecated methods is not guaranteed, and you should avoid using them. This happens very frequently and is one of the most frustrating aspects of working in Android – one day you learn how to use a complex Android class, and the next day a new version of the API is released, and much of what you just learned is deprecated. When this happens, you will need to consult the Android documentation to see how the desired functionality is now implemented. In this case, unfortunately, the new process to access a colour constant is quite complex, so we will defer it until later in the course.

Fortunately, in our practical task, we are going to be accessing string constants, and the Resources.getString(int id) method has not undergone big changes in the most recent versions of the API.

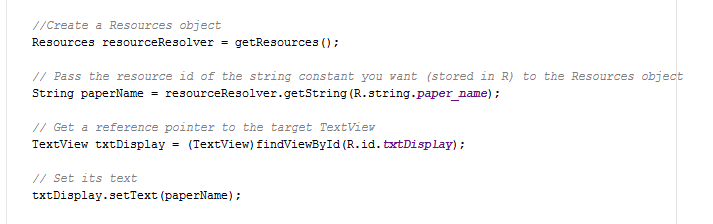
Assume we have modified /res/values/strings.xml as follows:



We can fetch the value of the paper\_name constant like this:



1. *Use the result.* Assume we wish to assign our string value to a TextView. As before, we use findViewById to get a reference to the control, and use the control’s setText method to change its contents. Thus, the complete code necessary to fetch and display a string resource constant is:



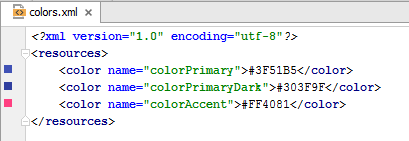
# Task: Work with Complex Resources

### Background:

The legal Android resource constant types are:[[1]](#footnote-1)

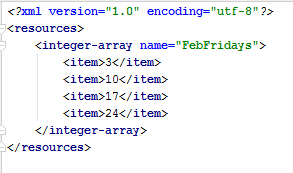
* <string>
* <dimen>
* <color>
* <bool>
* <integer>
* <integer-array> (a collection of integer <item>)
* <array> (a collection of any <item>)

We saw in lecture that the XML for defining the simple resource constants follows this pattern:



*Please note the North American spelling of "color". You can name your resources anything you like (e.g. name=”colourPrimary” would be perfectly acceptable), but the XML element tag must be <color>, without the u.*

**Array** **constants** have a more complex structure. To define an integer array resource, use the following XML format (we will see how to create the appropriate XML file in a moment):



To define general arrays, use the following XML format (example from android.com):

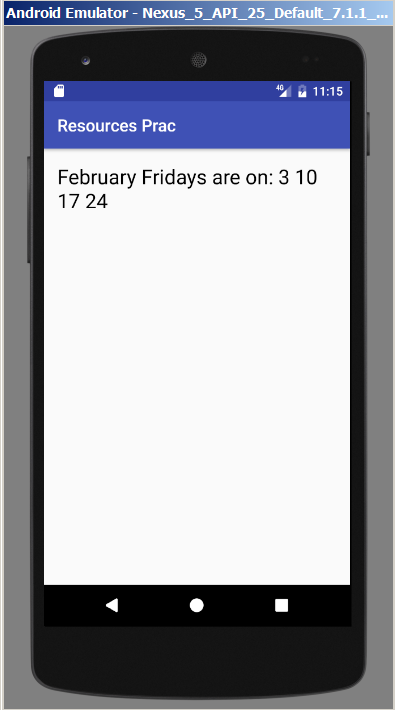


### Creating a New Resource File

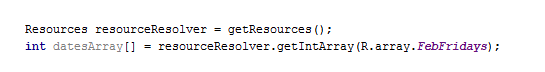
Keep your resources organised into sensibly named files. Resource files can be named whatever you want -- it is the XML tag names that must follow the rules above. To add a new resource file, right-click on the /res/values folder and select New->Values Resource File.

# Task:

* Add the FebFridays integer-array (shown above) to your current project as a resource in file res/values/integers.xml. You will need to create the file first.
* Modify your onCreate method so that, when the app launches, it reads and displays the contents of the array, like this:



* To access the contents of an <integer-array> resource constant, use a Resources object, calling its getIntArray(int id) method. This method returns an array of int. As an example, my solution contains these two lines of code:



* Make sure that your code is efficient. Iterate over the array of integers in a correctly expressed Java loop.

1. See https://developer.android.com/guide/topics/resources/more-resources.html for details. [↑](#footnote-ref-1)