**Adding dataBinding in an Activity:**

**USE THIS INSTEAD:-**

**https://developer.android.com/topic/libraries/view-binding**

Enable data binding in your build.gradle file in the app module inside the android section:

**dataBinding** {

**enabled** = true

}

Wrap all views in activity\_main.xml into a <layout> tag, and move the namespace declarations into the the <layout> tag.

In MainActivity, create a binding object:

**private** lateinit **var** binding: ActivityMainBinding

In onCreate, use DataBindingUtil to set the content view:

binding = DataBindingUtil.setContentView(**this**, R.layout.activity\_main)

Use the binding object to replace all calls to findViewById, for example:  
 binding.doneButton.setOnClickListener….etc

(Make sure to add an outer <layout> tag in the XML file)

**2. Create a fragment file.**

Make sure a file is selected in the project first.

Select File->New->Fragment->Fragment (Blank).

* For fragment name, use TitleFragment
* uncheck create Layout XML
* uncheck include fragment factory methods,
* uncheck include interface callbacks

Then select Finish.

**3. Cleanup the new TitleFragment class.**

Open up the new TitleFragment, begin by deleting the four lines under the import statements, from the // TODO. Rename parameter arguments, choose names that match, down to the ARG\_PARAM2 declaration. Then remove the return TextView(activity).apply section along with the SetText. The fragment won’t compile now because we have just removed the code that was returning the Fragment View or ViewGroup.

**4. Inflate and return the fragment layout.**

In our Activity, we used DataBindingUtil.setContentView to get the binding class from a layout, but since we’re in a fragment we need to call DataBindingUtil.inflate in onCreateView with the provided layout inflater, the layout resource ID, the provided viewgroup it will be hosted by, and false to not attach it to the viewgroup. Return binding.root. The code should look like:

*//Inflating and Returning the View with DataBindingUtil*

**override** fun onCreateView(inflater: **LayoutInflater**, container: **ViewGroup**?,savedInstanceState: **Bundle**?): **View**? {

**val** binding =**DataBindingUtil**.inflate<**FragmentTitleBinding**>(inflater, **R**.layout.fragment\_title, container, false)

**return** binding.root

}

**5. Add the fragment to the activity layout.**

Open the activity\_main.xml layout. Go to the LinearLayout and create a fragment tag inside of it. You need to give a fragment created in this way an id and set android:name to the full path of our fragment class. Then set the layout width and height to match\_parent, and you’re done.

**6. Run your app.**

Launch the app and revel in the glory of your first fragment!

### **1. Adding the Navigation Components to the Project**

We've already added the ext variable for you within the **Project** build.gradle file that shows the version of the navigation component we will use. We have tested against the version shown, but you can see which version is the latest on [this page on developer.android.com](https://developer.android.com/topic/libraries/architecture/adding-components):

buildscript {

ext {

**...**

version\_navigation = '1.0.0'

**...**

}

Within the **App** build.gradle file, add the dependencies for navigation fragment ktx and navigation UI ktx.

dependencies {

...

implementation "android.arch.navigation:navigation-fragment-ktx:$version\_navigation"

implementation "android.arch.navigation:navigation-ui-ktx:$version\_navigation"

}

### **2. Adding the Navigation Graph to the Project**

In the Project window, right-click on the res directory and select New > Android resource file. The New Resource dialog appears.

Select Navigation as the resource type, and give it the file name of navigation. Make sure it has no qualifiers. Select the navigation.xml file in the new navigation directory under res, and make sure the design tab is selected.

### **3. Replace the Title Fragment with the Navigation Host Fragment in the Activity Layout**

Go to the activity\_main layout. Change the class name of the existing Title fragment to androidx.navigation.fragment.NavHostFragment. Change the ID to myNavHostFragment. It needs to know which navigation graph resource to use, so add the app:navGraph attribute and have it point to the navigation graph resource - @navigation/navigation. Finally, set defaultNavHost = true, which means that this navigation host will intercept the system back key.

*<!-- The NavHostFragment within the activity\_main layout -->*

<fragment

android:id="@+id/myNavHostFragment"

android:name="androidx.navigation.fragment.NavHostFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

app:navGraph="@navigation/navigation"

app:defaultNavHost="true"

/>

### **4. Adding the Title and Game Fragments to the Navigation Graph**

Within the navigation editor, click the add button. A list of fragments and activities will drop down. Add fragment\_title first, as it is the start destination. (you’ll see that it will automatically be set as the Start Destination for the graph.) Next, add the fragment\_game.

*<!-- The complete game fragment within the navigation XML, complete with tools:layout. -->*

<fragment

android:id="@+id/gameFragment"

android:name="com.example.android.navigation.GameFragment"

android:label="GameFragment"

tools:layout="@layout/fragment\_game" />

### **5. Connecting the Title and Game Fragments with an Action**

Begin by hovering over the titleFragment. You’ll see a circular connection point on the right side of the fragment view. Click on the connection point and drag it to gameFragment to add an Action that connects the two fragments.

### **6. Navigating when the Play Button is Hit**

Return to onCreateView in the TitleFragment Kotlin code. The binding class has been exposed, so you just call binding.playButton.setOnClickListener with a new anonymous function, otherwise known as a lambda. Inside our lambda, use view.findNavcontroller to get the navigation controller for our Navigation Host Fragment. Then, use the navController to navigate using the titleFragment to gameFragment action, by calling navigate(R.id.action\_titleFragment\_to\_gameFragment)

*//The complete onClickListener with Navigation*

binding.playButton.setOnClickListener { view: View ->

view.findNavController().navigate(R.id.action\_titleFragment\_to\_gameFragment)

}

One more thing you might want to do. Navigation can create the onClick listener for us. We can replace the anonymous function with the Navigation.createNavigateOnClickListener call.

*//The complete onClickListener with Navigation using createNavigateOnClickListener*

binding.playButton.setOnClickListener(

Navigation.createNavigateOnClickListener(R.id.action\_titleFragment\_to\_gameFragment))

You’re done! That was a lot of steps, but you’ve built a graph, connected it to the UI, and made navigation happen

**ADDING ACTION BAR BACK BUTTON(UP BUTTON)**

**1. Link the NavController to the ActionBar with NavigationUI.setupWithNavController.**

Let's move to MainActivity. We need to find the NavController. Since we’re in the Activity now, we’ll use the alternate method of finding the controller from the ID of our NavHostFragment using the KTX extension function.

**val** navController = **this**.findNavController(**R**.id.myNavHostFragment)

Link the NavController to our ActionBar.

NavigationUI.setupActionBarWithNavController(**this**, navController)

**2. Override the onSupportNavigateUp method from the activity and call navigateUp in nav controller.**

Finally, we need to have the Activity handle the navigateUp action from our Activity. To do this we override onSupportNavigateUp, find the nav controller, and then we call navigateUp().

**override** fun onSupportNavigateUp(): **Boolean** {

**val** navController = **this**.findNavController(**R**.id.myNavHostFragment)

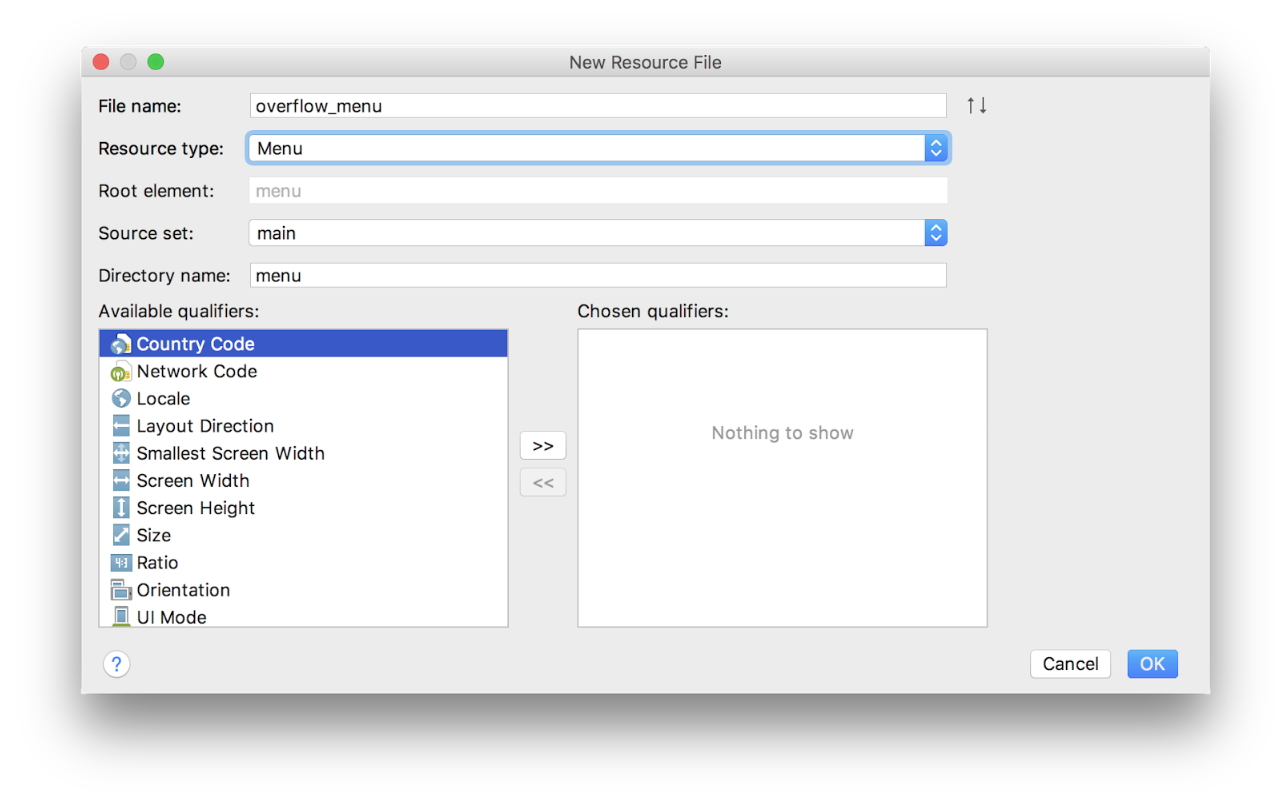
**return** navController.navigateUp()

}

**MENU SECTION**

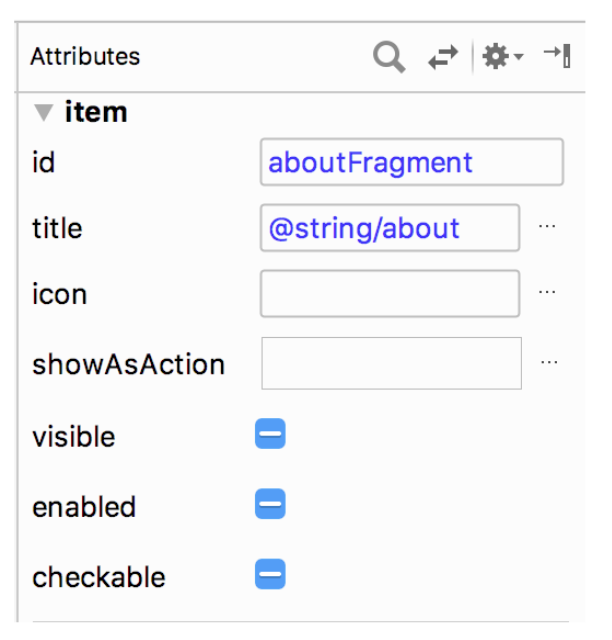
**2. Create new menu resource.**

Right click on the res folder within the Android project and select New Resource File. We’ll call this one overflow\_menu, with resource type of Menu. Click on the overflow\_menu within the menu directory, to view our new (empty) menu.



**3. Create “About” menu item with ID of aboutFragment destination**

Make sure the design tab is selected. Drag a menu item from the palette into the component tree below. Move to the attributes pane. Set the new item's id to aboutFragment, its destination. That's the id you used when adding the About fragment to the navigation graph. For title, we can use @string/about. The rest of the attributes should be left as their defaults.



**4. Call setHasOptionsMenu() in onCreateView of TitleFragment**

Next we need to tell Android that our TitleFragment has a menu. In onCreateView call setHasOptionsMenu(true).

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?): View? {

...

setHasOptionsMenu(true)

return binding.root

}

**5. Override onCreateOptionsMenu and inflate menu resource**

Next we need to override onCreateOptionsMenu and inflate our new menu resource using the provided menu inflater and menu structure.

override fun onCreateOptionsMenu(menu: Menu?, inflater: MenuInflater?) {

super.onCreateOptionsMenu(menu, inflater)

inflater?.inflate(R.menu.overflow\_menu, menu)

}

**6. Override onOptionsItemSelected and call NavigationUI.onNavDestinationSelected**

Finally, we need to override onOptionsItemSelected to connect it to our NavigationUI.

**override** fun onOptionsItemSelected(item: MenuItem?): Boolean {

**return** NavigationUI.onNavDestinationSelected(item!!,

view!!.findNavController())

|| super.onOptionsItemSelected(item)

}

And that’s it. Run the app and enjoy the "About" feature we’ve just added to the title fragment.