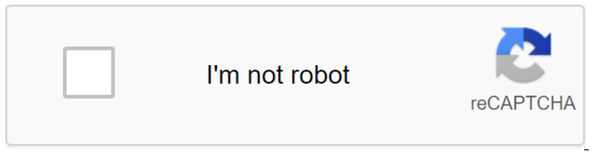
Fragment android example androidhive



Continue

An Android fragment is a GUI component that can live inside the Action. The Android fragment itself is not a View subclass, which are most of the other components of the GRAPHIC interface. Instead, the fragment has a view inside it. It is this view that is eventually displayed inside the activity in which the fragment lives. Because an Android snippet isn't a view, adding it to the action looks a little different than adding a view (such as TextView). The fragment is added to ViewGroup inside the action. A view of the snippet is displayed inside viewGroup. The following diagram shows what happens when a snippet is added to the action: first, the action gets a link to the fragment. He then gets a link to ViewGroup, the kind of snippet will be drawn inside. The action then adds a snippet. The snippet then creates its view and returns it to action. The view is then inserted into the ViewGroup parent, and the snippet is alive. Create a snippet To create a snippet you need to do two things: create a Fragment class. Create a snippet of the XML file layout. Create a Fragment class to create a Fragment class, create a regular Java class that expands android.app.Fragment. Here's an example: import android.app.Fragment; MyFragment's public class expands the Fragment - onCreateView () This subclass snippet does nothing yet. MyFragment needs to override the method onCreateView inherited from the Fragment to create a view of the fragment that should be displayed inside the action to which the fragment is added. Here's an example of a piece of implementationCreateView() implementation: MyFragment's public class expands the fragment - @Override a public view ofCreateView (LayoutInflater inflater, ViewGroup parentViewGroup, Bundle savedInstanceState) - View rootView - inflater.inflate inflate. (R.layout.fragment_my, parentViewGroup, false); Reverse rootView; The onCreateView method receives LayoutInflater, ViewGroup and Bundle as parameters. LayoutInflater is a component that can create a view copy based on the XML layout files. As you can see, the example actually does this by calling layout.inflate.com. The inflating method requires three parameters: the XML layout file ID (inside the R.layout), the parent ViewGroup, which should insert the view of the fragment, and the third boolean, telling you whether to insert a view of the piece as inflated from the XML layout file in the parent ViewGroup. In this case, we will take the false because the View will be attached to the parent ViewGroup elsewhere, some Android code that we call (in other words, behind our backs). When you pass a false as the last option to inflate (), parental ViewGroup is still used for calculations inflated submission, so you can't pass zero like Parental ViewGroup. ViewGroup onCreateView is the parent ViewGroup that should insert the view of the fragment. Fragment. is ViewGroup inside the action, which will be the master of the fragment. The onCreateView kit option is a kit in which a piece can store information, just like an action. Add a snippet to the action To show the view of the piece inside the action you need to add a snippet to the action. You do it inside the event. Here's an example that adds a snippet from inside onCreate activity: the myActivity community class expands activity - @Override protected void onCreate (Bundle savedInstanceState) - super.onCreate (savedInstanceState); setContentView (R.layout.activity_my); If (savedInstanceState - null) - getFragmentManager () .beginTransaction () .add (R.id.fragmentParentViewGroup, new MyFragment ().commit (); The fragment is added inside the if-statement. Unlike other View components, the action remembers which pieces were added to the action. Therefore, you should only add a snippet to the action once during the entire life of the operation, or the snippet will appear multiple times. This is true even if the user changes the orientation of the device between the portrait and the landscape, and even if the action is destroyed in the process. Android will remember that a snippet has been added to this activity when new activities of this kind are created, and will re-attach the snippet to the activity. If savedInstanceState is zero, then this is the first time any state is created for this activity, so the snippet hasn't been added to it yet. Keep in mind that if the app is completely destroyed by Android, it's savedinanceState Bundle variable, and then you need to reattached the snippet when the activity is up. Fragments must be added (or replaced or removed) inside FragmentTransaction. You get FragmentTransaction through FragmentManager. You get FragmentManager using the getFragmentManager method() from FragmentManager, you can get FragmentTransaction by calling the StartTransaction method. On FragmentTransaction, returned startTransaction, you can now add, replace, or remove the fragments into action. This is a method of adding () to FragmentTransaction, which adds a snippet to the activity. The addition method takes two parameters. The first option is the parent ViewGroup ID, which should insert the view of the fragment. The second option is a copy of the piece to add. When you've finished adding, replacing, or deleting snippets, you have to do FragmentTransaction. You do this by causing a commit () method of fragmentTransaction. Only after FragmentTransaction has been made will all the changes you make take effect. Here's a mock-up of the XML Action file. Note that the root item ID is the same as that of the link in the permanent collection <FrameLayout xmlns:android xmlns:android android:id'y id/fragmentParentViewGroup android:layout_widthmatch_parent Android:layout_height match_parent tools: context . MyActivity Tools:ignoreMergeRootFrame / This FrameLayout element is used as a parent viewGroup for a snippet. Thus, in practice, the fragment will take up the entire screen. Another ViewGroup could be used as a parent for the piece. Here's an example of the XML file, which shows how to zlt;LinearLayout xmlns:android/ xmlns:tools/ android:layout_width'match_parent android:layout_height'match_parent android:vertical'orientation tools:context. MyActivity'gt; TextView android:text'Activity Title android:layout_width'match_parent android:layout_height'wrap_content/textView LinearLayout android:id'id/fragmentParentViewGroup android:layout_width match_parent android:layout_height'match_parent android: orientation The fragment will use this LinearLayout as the parent viewGroup. The life Cycle Android fragment has a life cycle that is similar to the life cycle of activity. This illustrates the life cycle of the fragment: first, a snippet is added to the activity. This begins the life cycle of the fragment. Secondly, onAttach methods(), onCreate, onCreateView, onActivityCreated (), onStart and onResume are called. OnActivityCreated is called when hosting activities are fully established. If a snippet is removed from the hosting or the snippet is moved to the app's background mode (the other action moves to the foreground), the methods onPause (), onStop, and onDestroyView are called. If the snippet returns to visibility, the fragment can go from onDestroyView to onCreateView and become visible again. Replace a snippet you can replace a piece added to the action with another piece. You do this with a replacement method at FragmentTransaction. Here's an example: getFragmentManager ().startTransaction (R.id.fragmentParentViewGroup, new MySecondFragment ().(commit); You can replace the fragments while the activity is still alive. Once the action is destroyed, you can no longer replace the fragments inside it. You can replace the fragments as many times as you want. You can replace snippets as a way to change part of the user interface, or the entire user interface if that's what you need/prefer. You can remove a snippet from the action with FragmentTransaction. You do this with the removal method. Here's FragmentTransaction .commit(); The deletion method takes one option: a link to a deletion piece. The fragment will be removed from any parent ViewGroup, and a snippet has been added to it. By adding a piece of transaction to the back stack, you can add a piece transaction to the back stack. The back stack tracks the actions in your app that can be backed up when a user presses the standard Android Back button on the device. If you add a piece transaction to the back stack, the transaction may be receded (back) at the touch of the Back button on the device. Here's an example showing how to add a piece transaction to the back stack: MySecondFragment mySecondFragment - the new MySecondFragment getFragmentManager ().beginTransaction() .replace (R.id.fragmentParentViewGroup, mySecondFragment) .addToBackStack (null) .commit(); This example replaces any piece that has already been added to the parent view group, id R.id.fragmentParentViewGroup with a copy of MySecondFragment. The example also adds a replacement transaction to the back stack. The addToBackStack takes an additional line option that can determine this state in the back stack. Most of the time you won't need this option, so passing zero is enough. When this replacement transaction has been made, it can be cancelled by the user by clicking the Android device's Back button, in the same way the Back button can return the user to previously visible activity. Activities. viewpager in fragment android example androidhive. android fragment example androidhive