Fragment transaction dialog android

blob: e4c84d7e79779713698c5abb0d9eaeb7848a698c95 (file) (file) (fault) (C) (C) 2010 Android Open Source Project - Apache License, version 2.0 (license); You cannot use this file except under the License. You can obtain a copy of the License by phone: - Unless applicable law or agreed in writing, the software distributed under the License is covered by AS IS BASIS - WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. Cm. A license for a specific language regulating permission and restrictions under the License. Issue/package android.app;import android.compat.annotation.UnsupportedAppUsage;import android.content.Context;import android.content.DialogInterface;import android.os.Bundle;import android.view.layoutInflater;import android.view.View;import android.view.ViewGroup;import android.view.Window;import android.view.WindowManager;import java.io.FileDescriptor;import java.io.PrintWriter;/- Fragment that displays the dialogue window, floating on top of the window of his activity. This snippet contains a Dialog object that it displays as needed depending on the state of the fragment. Dialogue management (decision on when to show, hide, dismiss it) should be done through the API here, not with direct calls to dialogue. Implementations should override this class and implement @link #onCreateView (LayoutInflater, ViewGroup, Bundle) to deliver the content of the conversation. They can also override @link #onCreateDialog (Bundle) to create a fully user-like dialogue, such as AlertDialog, with its own content. Topics covered here #Lifecycle are: #BasicDialog't-a-basic #DialogOrEmbed #AlertDialog Dialogue The choice between dialogue or embedding is a choice between dialogue or embedding. DialogFragment does various things to save and manage the life cycle of the fragment, not the Dialogue. Note that dialogues are usually self-contained entities - they are their own window, getting their own input events, and often decide when to disappear (by receiving a back key event or pressing a button). DialogFragment must ensure that what happens to the Dialog fragment and states remains consistent. To do this, he monitors the dismissal of events from the dialogue and takes care of the removal of his own condition when they occur. means that you should use @link #show (FragmentManager, String) or @link #show (FragmentTransaction, String) to add a copy of DialogFragment to your user interface, as they track how DialogFragment should delete itself when the conversation is rejected. The basic Dialogue The simplest use of DialogFragment is a floating container for the hierarchy of fragment views. A simple implementation may look like this: Development/samples @sample/ApiDemos/src/com/example/android/apis/app/FragmentDialog.java add_dialog @sample When the transaction is jumped, the current DialogFragment and its Dialog will be destroyed, and the previous (if any) re-shown. Please note that in this case, DialogFragment will take care to push out the Dialogue transaction and shrug it off separately. The name of AlertDialog Instead of (or in addition to) implementing (@link #onCreateView) to create a hierarchy of views within the dialogue, you can create a hierarchy of views within the dialogue, you can (@link #onCreateDialog (Bundle) to create your own Dia @link log custom object. A simple example of this is: @sample development/ samples / ApiDemos/src/com/example/android/apis/app/FragmentAlertDialog.java methods to show the dialogue and get results from it: @sample development/samples/ApiDemos/src/com/example/android/apis/app/FragmentAlertDialog.java It's just being added as an endlessly running piece. Because the conversations are usually modal, this will still work as a back stack, as the dialogue will capture the user input until it is rejected. When he is fired, DialogFragment will take care of removing himself from its manager snippets. The choice between dialogue or embedding can still be used as a regular piece if desired. This is useful if you have a snippet that in some cases should be shown as a dialogue while others are built into a larger user interface. This behavior will usually be automatically selected for you depending on how you use the snippet, but can be configured with a @link #setShowsDialog boolean. For example, here's a simple snippet of the dialogue: @sample development/samples/apiDemos/src/com/example/android/apis/app/FragmentDialogOrActivity.java @sample/ApiDemos/src/com/example/android/apis/app/FragmentDialogOrActivity.java show_dialog hierarchy: @sample development/samples/ApiDemos/src/com/example/android/apis/app/FragmentDialogOrActivity.java - @deprecated Use the library of the @docRoot/@docRoot/extras/support-library.html support @link android.support.v4.app.DialogFragment for consistent behavior on all devices and access to the @docRoottopic/libraries/architecture/lifecycle.@Deprecatedpublic html DialogInterface.OnDismissListener (. . . . . STYLE_NORMAL @link #setStyle STYLE_NO_TITLE @link #setStyle @link #setStyle . . . . . the view hierarchy returned to @link #onCreateView @link #setStyle STYLE_NO_FRAME (who is fully responsible for drawing the dialogue. @link #STYLE_NO_FRAME, but also disables all inputs in the SAVED_DIALOG_STATE_TAG STYLE_NO_INPUT dialogue. :preservedDialogState; private static final string SAVED_STYLE - android:style; private static final string SAVED_THEME and android:theme; private static final string SAVED_CANCELABLE and android:cancelled; private static final string SAVED_SHOWS_DIALOG - android:showsDialog; private static final string SAVED_BACK_STACK_ID and android:backSt STYLE_NORMAL ed int mTheme No 0; boolean mCancelable - truth; boolean mShowsDialog - true; @UnsupportedAppUsage int mBackStackId -1; mDialog Dialogue; @UnsupportedAppUsage boolean mViewDeDe Stroyed; @UnsupportedAppUsage mDismissed; @UnsupportedAppUsage mShownByMe; public DialogFragment () - call to customize the main appearance and behavior of the dialogue fragment. as well as to choose flags, themes and other options for you. The same effect can be achieved by manually installing Dialog and Window, attributes themselves. Calling this after creating the Fragment Dialogue will have no effect. This @param chooses a standard style: it can be @link #STYLE_NORMAL, @link #STYLE_NO_TITLE, @link #STYLE_NO_FRAME or @link #STYLE_NO_INPUT. - @param Theme Extra custom theme. If 0, the appropriate theme (based on style) will be chosen for you. In/ public emptiness setStyle (int style, int theme) - mStyle - style; if (mStyle - STYLE_NO_FRAME mStil - STYLE_NO_INPUT) - mTheme - com.android.internal.R.style.Theme_DeviceDefault_Dialog_NoFrame;to this FragmentManager. It's a convenience for explicitly creating a transaction, adding a snippet to it with that tag, and making it. This doesn't add to the back stack. When the snippet is removed, a new transaction will be executed to remove it from the action. - @param manager FragmentMenager this snippet will be added to. - @param tag tag for this snippet, according to @link FragmentTransaction (Fragment, String) FragmentTransaction.add. In/ public invalid show (FragmentManager manager, String tag) - mDismissed - false; mShownByMe - the truth; FragmentTransaction ft - manager.beginTransaction(); ft.add ft.commit(); (. . . @hide . . . . . @UnsupportedAppUsage . . . . . FragmentTransaction ft - manager.beginTransaction(); ft.add ft.commitLegingStateLoss (); Show the conversation, add a snippet using an existing transaction, and then make a transaction. - @param transaction existing transaction, in which you can add a fragment. - @param tag tag for this snippet, according to @link Fragment Fragmentaction (Fragment, String) FragmentTransaction.add. - @return returns the transaction ID, according to @link FragmentTransaction.commit () . - Public int show (FragmentTransaction transaction, String tag) - mDismissed - false; mShownByMe - the truth; transaction.add mViewDestroyed - false; mBackStackId - transaction.commit(); Return mBackStackId; If a snippet has been added to the backup stack back, the entire state of the stack is back up before and including this entry will be popped. Otherwise, a new transaction will be made to remove the piece. Dismissal from public void () - dismissalinternal (false); / / / Version @link #dismiss that uses @link FragmentTransaction CommitLegalingStateLoss () - FragmentTransaction.commitAllowingStateLoss ()). For more information, you can review related documentation. Issue/public dismissalDefinable StateLoss - dismissalInternal (admittedly); - invalid dismissal of the Eternal (Bulian permit StateLoss) - if (mDismissed) - return; mDismissed - the truth; mShownByMe - false; if (mDialog ! - null) - mDialog.dismiss (); mDialog - null; mViewDestroyed - the truth; If (mBackStackId zgt; x 0) - getFragmentManager ().popBackStack (mBackStackId, FragmentManager.POP_BACK_STACK_INCLUSIVE); mBackStackId -1; - more - FragmentTransaction ft - getFragmentManager (.beginTransaction (); ft.remove If (allow StateLoss) - ())) ; - still - ft.commit (); - Public Dialog getDialog () - return mDialog; - public int getTheme ( ) - return mTheme; Use this instead of a direct @link @link call - Dialog.setCancelable (boolean) because DialogFragment needs to change its behavior based on this. If @param is cancelled, the dialogue is cancelled. This is true by default. The public void established by Cancelable (boolean cancelable) - mCancelable - is abolished; @link #setCancelable if (mDialog !) null) mDialog.setCancelable (cancelable); B/public bulean is a smemage () - return mCancelable; If not a set, the dialogue will not be created in the @link #onActivityCreated (Bundle), and thus the hierarchy of the view of the piece will not be added to it. This allows you to use it instead as a normal piece (embedded inside its activities). This is usually set for you depending on whether the fragment is related to the container viewing ID transferred to @link FragmentTransaction add (int, Fragment) FragmentTransaction.add (int, Fragment). If a snippet was added with a container, setShowsDialog would be initiated into a false; otherwise, it will be true. - @param showsDialog If this is true, the snippet will appear in the Dialogue. If a lie, no Dialogue will be created, and the representation of the fragment hierarchically will not remain untouched. - public set of voidsShowsDialog (boolean showsDialog) - mShowsDialog @link #setShowsDialog - showsDialog; Issue/ public boolean getShowsDialog - return mShowsDialog; - @Override public void onAttach (Contextual Context) - super.onAttach (context); if (!mShownByMe) / If not explicitly shown through our API, take this as a / indication that the dialogue is no longer dismissed. mDismissed - false; - @Override public void on Detach () - super.onDetach (); if (!mShownByMe! mDismissed) / The fragment was not shown by a direct call here, it is not!) dismissed and now it detachs... Well, well, you' art is now fired. Have fun. mReport and true; - public @Override onCreate (Bundle savedInstanceState) - super.onCreate (savedInstanceState); mShowsDialog - mContainerId No 0; If (saved Of the life ! - null) - mStyle - savedInstanceState.getInt (SAVED_STYLE, STYLE_NORMAL); mTheme - savedInstanceState.getInt (SAVED_THEME, 0); mCancelable - savedInstanceState.getBoolean (SAVED_CANCELABLE, true); mShowsDialog - savedInstanceState.getBoolean (SAVED_SHOWS_DIALOG, mShowsDialog); mBackStackId @hide @Override - savedInstanceState.getInt (SAVED_BACK_STACK_ID, -1); savedInstanceState) - if (!mShowsDialog) - return super.onGetLayoutInflater (saved State); mDialog - onCreateDialog (savedInstanceState); Switch (mStyle) - case STYLE_NO_INPUT: mDialog.getWindow ().addFlags (WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE) WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE); Fall through ... STYLE_NO_FRAME case: STYLE_NO_TITLE case: - if (mDialog !) - return (LayoutInflater)mDialog.getContext (.getSystemService (Context.LAYOUT_INFLATER_SERVICE); - return (LayoutInflater) mHost.getContext (.getSystemService (Context.LAYOUT_INFLATER_SERVICE); Override to create your own custom Dialog container. This is usually used for the AlertDialog show instead of general Dialogue; @link #onCreateView (LayoutInflater, ViewGroup, Bundle) doesn't need implementation because AlertDialog cares about its own content. This method will be named after @link #onCreate (Bundle) and @link #onCreateView (LayoutInflater, ViewGroup, Bundle). Implementing by default simply instantly returns the @link Dialog class. Note: DialogFragment owns the @link Dialogue (Dialog)setOnCancelListener (Dialog.setOnCancelListener) and @link DialogsetOnDismissenLister - Dialog.setOnDississListener. @param @link #onDismiss @link #onCancel You don't have to install them yourself. - @return to return a new dialog instance that will be displayed by The Fragment. Issue/Public Dialogue onCreateDialog (Bundle savedInstanceState) - the return of the new Dialogue (getActivity(, getTheme (); - Public void on Cancel (DialogInterface dialog) - public void onDismiss (DialogInterface dialog) - if (!mViewDestroyed) -/ Note: we need to use allowStateLoss, because the dialogue // sends this asynchronous so we can get a call // after the action is suspended. The worst case is when the user comes in / Back to the activity they see the dialogue again. to dismiss the International (admittedly); - @Override void onActivityCreated (Bundle savedInstanceState) - super.onActivityCreated (preserved State); If (!mShowsDialog) - return; View view and getView If (see !' null) - if (view.getParent () ! - null) - throw the new IllegalStateException (DialogFragment can not be attached to the view of the container); If (activity ! - null) - mDialog.setOwnerActivity (activity); mDialog.setCancelable (mCancelable); If (!mDialog.takeCancelAndDismissListeners (DialogFragment, this is) - throw the new IllegalStateException (you can't install Dialog's OnCancelListener or OnDismissListener) SAVED_DIALOG_STATE_TAG; @Override public void on Start - super.onStart (); If (mDialog !) - mViewDestroyed - false; mDialog.show (); - @Override public void onAveInstanceState (Bundle outState) - super.onSaveInstanceState (outState); если (диалогГосударство! - null) - outState.putBundle (SAVED_DIALOG_STATE_TAG, dialogState); если (mStyle! - STYLE_NORMAL) - outState.putInt (SAVED_STYLE, mStyle)SAVED_THEME; mTheme); если (!mCancelable) - outState.putBoolean (SAVED_CANCELABLE, mCancelable); если (!mShowsDialog) - outState.putBoolean (SAVED_SHOWS_DIALOG, mShowsDialog)SAVED_BACK_STACK_ID; @Override публичная пустота наStop() - super.onStop (); если (mDialog !) - null) - mDialog.hide (); Вопрос/ @Override публичная пустота наDestroyView () - super.onDestroyView (); если (mDialog! - null) // Установить удалены здесь, потому что это увольнение только для того, чтобы скрыть // диалог - мы не хотим, чтобы это привести к фрагмент // на самом деле быть удалены. mViewDestroyed - правда; mDialog.dismiss (); mDialog - null; - @Override публичная свалка пустоты (String prefix, FileDescriptor fd, PrintWriter writer, String'args) - super.dump (префикс, fd, писатель, args); writer.print (префикс); writer.print(prefix); writer.println(DialogFragment); writer.print (префикс); writer.print (mStyle); writer.print (mTheme-0x); writer.println(Integer.toHexString(mTheme)); writer.print (префикс); writer.print (mCancelable) ; writer.print (mCancelable); writer.print (mShowsDialog); writer.print (mShowsDialog); writer.print (mBackStackId); writer.println(mBackStackId); writer.print (префикс); writer.print (МДиалог); writer.println (mDialog); writer.print (префикс); writer.print (mViewDestroyed); writer.print (mViewDestroyed); writer.print (mDismissed) ; writer.print (mDismissed); writer.print (mShownByMe); writer.println (mShownByMe); }} }}