About This Article

In this article we describe how to manage the activity stack when an application has more than one activity, and how to use notifications and pending intents

Scope:

This article is intended for users who want brief understanding on Android programming. This article guides you on developing Applications using Notifications and Activity History in Android. It assumes that the user has already installed Android and necessary tools to develop application. It also assumes that the user to be familiar with Java or familiar with Object Oriented Programming concepts.

Introduction

In this article we describe how to manage the activity stack when an application has more than one activity, and how to use notifications and pending intents. As an example, this article defines a music player in which we have three activities:

- A list of authors
- A list of songs (for a specific author)
- Details about the currently selected and/or playing song.

This application will also have a notification in a status bar to allow quick access to the player from any location. This will NOT be a functioning music player – nothing will be played, and we will not manage real mp3 files. This will be only a sample application that shows how to manage the stack of activities using intents. For this reason we create two classes that simulate two things: class Folders – to simulate real files and folders, and class CurrentInfo to keep info about the song currently playing. The Folder class has two important static fields:

- String AUTHORS[] - which contains names of all authors
- ArrayList<String> SONGS[] - which is a table of ArrayLists (every element of this table keeps an ArrayList)

Every ArrayList represents one specific author and contains a list of this author's songs. In your own music player you should replace this class with a real file manager, which will keep info about the authors and songs you have. It doesn't have to keep this info in exactly that way, this is just an example. The class CurrentInfo contains ten static fields – they keep information about currently selected and currently playing song. You should also replace this class, and implement your own service for playing music.

Flags in Intents

There are a few important things about intents. First, you should know how intent flags work. In the example application there is only one instance of each activity – especially of the activity that shows the song currently playing. Second, we want to show an already existing activity rather than closing and re-creating it. The simplest way to do that is to set the FLAG_ACTIVITY_REORDER_TO_FRONT flag for the intent.

```
Intent intent = new Intent(getApplicationContext(), Playing.class);
intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
```

```
startActivity(intent);
```
If this flag is set, and the intent wants to display the activity that is already running in this task; or, in other words, is running somewhere in the history stack of this task; then new activity will not be launched – instead the existing one will be brought to the front. For example, if we have a stack of four Activities: A, B, C, D. When D sends an intent that calls activity B, then activity B will be brought to the front and the new stack will be: A, C, D, B.

This doesn’t always work. For example, when your application isn't visible on the screen (the application is running in the background) and you send this intent through the notification (notice that it doesn't matter what activity of your application is brought to the front – it's enough that any of these activities are visible on the screen, and the notification's intent will work properly). It happens because the activity called by the notification's intent will start outside of the context of the existing activity. Therefore, it will automatically set up the FLAG_ACTIVITY_NEW_TASK flag. In this case you get a completely new activity rather than the old one brought to the front. In some cases this wouldn't be a problem, but this time we want to keep our old activity. One way to solve this problem is to set up two flags in the pending intent: FLAG_ACTIVITY_CLEAR_TOP, and FLAG_ACTIVITY_SINGLE_TOP.

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)getSystemService(ns);

int icon = android.R.drawable.ic_media_play;
CharSequence showText = "Activity Stack - Notification";
long time = System.currentTimeMillis();
Notification notification = new Notification(icon, showText, time);

Context context = Playing.this;
CharSequence contentTitle = "Currently Playing:";
CharSequence contentText = “author - title”;
Intent notifIntent = new Intent(getApplicationContext(), Playing.class);
notification.flags = Notification.FLAG_ONGOING_EVENT;
notification.flags |= Notification.FLAG_NO_CLEAR;

notifIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP|    Intent.FLAG_ACTIVITY_SINGLE_TOP);
PendingIntent contentIntent = PendingIntent.getActivity(context, 0, notifIntent,        0);
notification.setLatestEventInfo(context, contentTitle, contentText,     contentIntent);
mNotificationManager.notify(Playing.NOTIF_ID, notification);
```
This provides an instance of the activity that is already open (this is what we want), but the FLAG_ACTIVITY_CLEAR_TOP flag will close all of the activities ahead in the stack.

So let's get back to our example: we have stack with A, B, C, D, but our application is in the background. Pending intent calls activity B, activities C and D will be closed, and after that new stack will be: A, B. that’s not the only problem – after this Intent, pushing the BACK button won't show activity A. Instead, we will see either the home screen or the application that was open when we clicked on the notification. That's because the BACK button will navigate to the most recent open activity. There is a solution for that, and that is to override the onKeyDown(...) method for the BACK button. This method shouldn't bring us back to the previous activity, but should instead open activity

D, because it was on the top of the stack before we ran the notification's intent.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)  {
         if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
         // do something on back.
         Intent intent = new Intent(getApplicationContext(), MainActivity.class);
         startActivity(intent);
         return true;
         }
return super.onKeyDown(keyCode, event);
}
```

Sending Information Between Activities

Now let's get back to our “music player”. We've got three activities there:

- The main activity – with the list of authors
- The second activity – with the list of songs for each author
- The playing activity – with some details about the currently playing and selected songs (we assume that those can be two different songs).

The main activity and the second activity are simply list views. After choosing the author in the main activity we should see the second activity with a list of songs from this author. This should be done using an intent.

We also have to pass info to the second activity about a chosen author. There are two ways to do this: we can pass some extra data about the selected author in an intent using the putExtra(...) method, or we can save info in some additional variable – for that we use a separate class CurrentInfo. In this example application there is no risk of simultaneous writing to this class, but you should keep in mind that something like that could happen in a real player. Passing info by an intent using putExtra(...) has this advantage, that you don't have to worry about synchronizing. But you have to remember to always keep this activity alive, or save the information in some other place (like another class or activity) – otherwise you might lose info about a selected author when you exit. It is very likely that you will need additional classes or variables to pass and keep the data even if you want to use the putExtra(...) method. In our sample application we use the CurrentInfo class to pass the data between activities.

When you see the activity with a list of songs, you can select any song. Then you should see the application playing. You should pass info about the selected song to this activity the same way as before. The playing activity can be started only with a proper intent. In this activity, click the PLAY or STOP buttons, to play or stop the song you've just selected (again, nothing is really being played or stopped in our application).

Notice that when you push the BACK button the playing activity will be destroyed. We want to avoid this, so we override the BACK button action. The new action for this button is to run the proper intent to bring the second activity to the front (to the top of stack). We can do this by setting the FLAG_ACTIVITY_REORDER_TO_FRONT flag for this intent.

```
@Override
```

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
                // do something on back.
                Intent intent = new Intent(this, SecondActivity.class);
                intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
                startActivity(intent);
                return true;
        }
        return super.onKeyDown(keyCode, event);
}
```

Remember, if you pass info using putExtra(...), it doesn't matter if you run the second activity from the main activity or the playing activity. The same information is sent either way. This extra data acts like parameters (by this extra data you identify which list of songs should be shown on the screen) – and if you do not pass any extra data then you won't choose a list of songs to show and application may crash. This is the main advantage of using a separate class (CurrentInfo class in this case) to keep info about currently selected/playing authors and songs – you don't have to pass this info – the only thing you have to remember is to update this data when the user makes new selections.

Notifications

The next thing we want to describe is changing (or updating) notifications. In our player we want a notification to always be up-to-date. It means that every time we push the HOME button and go to the home screen (or run other applications), and later if we click on the player's notification we should go back to exactly the same place in the player we were before. Every notification has an unique ID – unique only within the namespace of your application's package – so if you have only one notification in your application (like in this one) then you can choose any integer (we chose 1). You can put this integer explicitly in your code, but it's good practice to put the integer as a static final value – in case you want to upgrade your application in the future, because it makes source code easier to read and analyze.

To update your notification you have to add a FLAG_CANCEL_CURRENT flag to the pending intent. This flag means that if the pending intent is already defined then the current one is canceled before the new one is created. Other pending intents' flags won't be used in this application. We should also set proper flags for the notification itself - FLAG_ONGOING_EVENT and FLAG_NO_CLEAR. The first flag should be set if the notification is in reference to some ongoing actions – like playing music. The second flag should be set when the notification shouldn't be removed from the notification bar when user clicks on the CLEAR button. Creating the notification should look like this:

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)getSystemService(ns);

int icon = android.R.drawable.ic_media_play;
CharSequence showText = "Activity Stack - Notification";
long time = System.currentTimeMillis();
Notification notification = new Notification(icon, showText, time);

Context context = getApplicationContext();
CharSequence contentTitle = "Link Second Activity";
```

```
CharSequence contentText = "Nothing is playing...";
Intent notifIntent = new Intent(this, SecondActivity.class);
notification.flags = Notification.FLAG_ONGOING_EVENT;
notification.flags |= Notification.FLAG_NO_CLEAR;

notifIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP|    Intent.FLAG_ACTIVITY_SINGLE_TOP);
PendingIntent contentIntent = PendingIntent.getActivity(context, 0, notifIntent,
PendingIntent.FLAG_CANCEL_CURRENT);
notification.setLatestEventInfo(context, contentTitle, contentText,     contentIntent);
FmNotificationManager.notify(Playing.NOTIF_ID, notification);
```

Checking How Switching Between Activities Works

In our sample application we also add an “edit text” view to the bottom of every activity screen. This field shouldn't appear in a real player, but we add it here for a specific purpose. When you're creating a new activity the edit text view should be empty. When you type something in this text view, or, in the main activity; then go to any other activity and then go back to this (main) activity you should see the text you've typed before. If you don't see it, it means that it is another instance of this activity. In case of the main and the second activities this is not a problem. Firstly, because these activities don't do anything in the background; and secondly, all the info needed to re-create these activities is in the CurrentInfo class. However, in the case of the playing activity we cannot simply re-create it, because it may do some actions in the background (it may play the music), or it may communicate with some services (services may play the music) and when we close this activity then the music probably will stop and we'll lose our “progress” in playing. That's why we have to make sure that the playing activity won't be closed. Because of that we add an async task (that shows simple progress bar) to our playing activity so we can easily check if it's a new instance of this activity or the instance that has already been working for some time. This progress bar is added also to simulate playing a song.

Resuming Activities

Usually when you develop an application you create and initiate all GUI-connected things in the onCreate(...) method of your activity class. However, in the case of an application in which you can “jump” between activities, this is not all you should do, especially when you don't want to close the activity each time you leave it, and then re-create it when you go back to it. It's because when you don't close the activity when you're leaving it (it still exists in the history stack) and later you're back to this activity the onCreate(...) function won't be called and the GUI won't be refreshed.

For example:

- You run the music player and the main activity (let's call it M) is on top now.
- Then you choose an author and you're going to the second activity (with list of songs – let's call this activity S), but M is still running – so the activity stack looks like this: M, S.
- Then you choose a song and you see the playing activity (let's say activity P). So now the history stack is: M, S, P. You click the PLAY button – and from now on this activity shouldn't be closed (for the reasons described above).
- Then you click the BACK button and you go back to the S activity. The history stack is now: M, P, S.
- Then you choose another song, click it and now you see the playing activity exactly the same as

it was before.

You didn't actually choose any new song. It's because the activity P was already created, and this time the only thing that happened was bringing this activity back to the top (FLAG_ACTIVITY_REORDER_TO_FRONT was set for the intent). To avoid this situation you have to override the onResume() method for this activity. In this situation onResume() should do almost everything onCreate(...) does – except for initiating all of the global variables. This will cause that the playing activity will be refreshed every time it is brought to the front. It is safe to override onResume() methods also for the other activities in this application. It's possible that these activities won't be closed and will need to be refreshed after you bring it to the top. You should remember that the onResume() method needs to call to the super class's implementation (super.onResume()). In the onResume() method you should also refresh the notification – probably the info (title, icon, text) and the pending intent should be updated. Below you can see the onResume() method for playing activity.

```
@Override
public void onResume() {
   super.onResume();
   TextView textViewSelected = (TextView)
   this.findViewById(R.id.textViewSelected);
   textViewSelected.setText(CurrentInfo.currentSelectedAuthor +
                  " : " + CurrentInfo.currentSelectedTitle);

   // Creating (updating) notification or updating variables
   if (CurrentInfo.currentPlayingTitleId == -1) { // check if something
       // is playing or not
     updateNotification();
   } else {
     CurrentInfo.tempSelectedAuthor =
     CurrentInfo.currentSelectedAuthor;
     CurrentInfo.tempSelectedAuthorId =
     CurrentInfo.currentSelectedAuthorId;
                 }
}
```

Update notification method for the playing activity:

```
private void updateNotification() {
         String ns = Context.NOTIFICATION_SERVICE;
         NotificationManager mNotificationManager = (NotificationManager)
                  getSystemService(ns);

         int icon = android.R.drawable.ic_media_pause;
         CharSequence showText = "Activity Stack - Notification";
         long time = System.currentTimeMillis();
         Notification notification = new Notification(icon, showText, time);
```

```
        Context context = this;
        CharSequence contentTitle = "Link Playing Activity";
        CharSequence contentText = "Nothing is playing...";
        Intent notifIntent = new Intent(this, Playing.class);
        notification.flags = Notification.FLAG_ONGOING_EVENT;
        notification.flags |= Notification.FLAG_NO_CLEAR;

        notifIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
                Intent.FLAG_ACTIVITY_SINGLE_TOP);
        PendingIntent contentIntent = PendingIntent.getActivity(context, 0,
                notifIntent,
        PendingIntent.FLAG_CANCEL_CURRENT);
        notification.setLatestEventInfo(context, contentTitle, contentText,
                contentIntent);
        int notif_ID = 1;
        mNotificationManager.notify(Playing.NOTIF_ID, notification);
}
```

Changing orientation of the screen from portrait to landscape or back calls the onCreate(...) method. We don't want to re-create activities (especially the playing activity). To avoid this we need to add one parameter in android manifest file in activity section:

```
android:configChanges="orientation|keyboardHidden"
```

Or you can just lock screen orientation. Add this line to android manifest file:

```
android:screenOrientation="landscape"
```

Updating Notifications

Finally, updating notifications should be done in a reasonable way. This means that unless nothing is playing, the notification should be updated every time the user switches to another activity. So every time we go to the home screen or any other application from our application, we can go back to the last activity we were using before we left our “player” application by clicking on the notification. When nothing is playing then clicking on the notification should take you back to exactly the same place in the application we were last time. However, if something IS playing, then it's a good practice to make sure that clicking on the notification will always bring us back to the playing activity – no matter what activity we saw last time. This is because it makes it possible to stop or change the music quickly. Updating notifications should depend on whether or not there's something currently being played.

Ref:http://developer.samsung.com/android/technical-docs/Notifications-and-Activity-History