Providing Proper Back Navigation

Back navigation is how users move backward through the history of screens they previously visited. All Android devices provide a *Back* button for this type of navigation, so **your app should not add a Back button to the UI**.

In almost all situations, the system maintains a back stack of activities while the user navigates your application. This allows the system to properly navigate backward when the user presses the *Back* button. However, there are a few cases in which your app should manually specify the *Back* behavior in order to provide the best user experience.

> **Back Navigation Design**
>
> Before continuing with this document, you should understand the concepts and principles for *Back* navigation as described in the Navigation (/design/patterns/navigation.html) design guide.

THIS LESSON TEACHES YOU TO:

1. Synthesize a new Back Stack for Deep Links
2. Implement Back Navigation for Fragments
3. Implement Back Navigation for WebViews

YOU SHOULD ALSO READ

- Providing Ancestral and Temporal Navigation
- Tasks and Back Stack
- Android Design: Navigation

Navigation patterns that require you to manually specify the *Back* behavior include:

- When the user enters a deep-level activity directly from a notification, an app widget, or the navigation drawer.
- Certain cases in which the user navigates between fragments.
- When the user navigates web pages in a `WebView`.

How to implement proper *Back* navigation in these situations is described in the following sections.

Synthesize a new Back Stack for Deep Links

Ordinarily, the system incrementally builds the back stack as the user navigates from one activity to the next. However, when the user enters your app with a *deep link* that starts the activity in its own task, it's necessary for you to synthesize a new back stack because the activity is running in a new task without any back stack at all.

For example, when a notification takes the user to an activity deep in your app hierarchy, you should add activities into your task's back stack so that pressing *Back* navigates up the app hierarchy instead of exiting the app. This pattern is described further in the Navigation (/design/patterns/navigation.html#into-your-app) design guide.

Specify parent activities in the manifest

Beginning in Android 4.1 (API level 16), you can declare the logical parent of each activity by specifying the `android:parentActivityName` (/guide/topics/manifest/activity-element.html#parent) attribute in the `<activity>` (/guide/topics/manifest/activity-element.html) element. This allows the system to facilitate navigation patterns because it can determine the logical *Back* or *Up* navigation path with this information.

If your app supports Android 4.0 and lower, include the Support Library (/tools/extras/support-library.html) with your app and add a `<meta-data>` (/guide/topics/manifest/meta-data-element.html) element inside the `<activity>` (/guide/topics/manifest/activity-element.html). Then specify the parent activity as the value for `android.support.PARENT_ACTIVITY`, matching the `android:parentActivityName` (/guide/topics/manifest/activity-element.html#parent) attribute.

For example:

```
<application ... >
    ...
    <!-- The main/home activity (it has no parent activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
```

```
        ...
    </activity>
    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- The meta-data element is needed for versions lower than 4.1 -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

With the parent activity declared this way, you can use the `NavUtils` (/reference/android/support/v4/app/NavUtils.html) APIs to synthesize a new back stack by identifying which activity is the appropriate parent for each activity.

Create back stack when starting the activity

Adding activities to the back stack begins upon the event that takes the user into your app. That is, instead of calling `startActivity()` (/reference/android/content/Context.html#startActivity(android.content.Intent)), use the `TaskStackBuilder` (/reference/android/support/v4/app/TaskStackBuilder.html) APIs to define each activity that should be placed into a new back stack. Then begin the target activity by calling `startActivities()` (/reference/android/support/v4/app/TaskStackBuilder.html#startActivities()), or create the appropriate `PendingIntent` (/reference/android/app/PendingIntent.html) by calling `getPendingIntent()` (/reference/android/support/v4/app/TaskStackBuilder.html#getPendingIntent(int, int)).

For example, when a notification takes the user to an activity deep in your app hierarchy, you can use this code to create a `PendingIntent` (/reference/android/app/PendingIntent.html) that starts an activity and inserts a new back stack into the target task:

```
// Intent for the activity to open when user selects the notification
Intent detailsIntent = new Intent(this, DetailsActivity.class);

// Use TaskStackBuilder to build the back stack and get the PendingIntent
PendingIntent pendingIntent =
        TaskStackBuilder.create(this)
                        // add all of DetailsActivity's parents to the stack,
                        // followed by DetailsActivity itself
                        .addNextIntentWithParentStack(upIntent)
                        .getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(pendingIntent);
...
```

The resulting `PendingIntent` (/reference/android/app/PendingIntent.html) specifies not only the activity to start (as defined by `detailsIntent`), but also the back stack that should be inserted into the task (all parents of the `DetailsActivity` defined by `detailsIntent`). So when the `DetailsActivity` starts, pressing *Back* navigates backward through each of the `DetailsActivity` class's parent activities.

> **Note:** In order for the `addNextIntentWithParentStack()` (/reference/android/support/v4/app/TaskStackBuilder.html#addNextIntentWithParentStack(android.content.Intent)) method to work, you must declare the logical parent of each activity in your manifest file, using the `android:parentActivityName` (/guide/topics/manifest/activity-element.html#parent) attribute (and corresponding `<meta-data>` (/guide/topics/manifest/meta-data-element.html) element) as described above.

Implement Back Navigation for Fragments

When using fragments in your app, individual FragmentTransaction (/reference/android/app/FragmentTransaction.html) objects may represent context changes that should be added to the back stack. For example, if you are implementing a master/detail flow (descendant.html#master-detail) on a handset by swapping out fragments, you should ensure that pressing the *Back* button on a detail screen returns the user to the master screen. To do so, call addToBackStack() (/reference/android/app/FragmentTransaction.html#addToBackStack(java.lang.String)) before you commit the transaction:

```
// Works with either the framework FragmentManager or the
// support package FragmentManager (getSupportFragmentManager).
getSupportFragmentManager().beginTransaction()
                           .add(detailFragment, "detail")
                           // Add this transaction to the back stack
                           .addToBackStack()
                           .commit();
```

When there are FragmentTransaction (/reference/android/app/FragmentTransaction.html) objects on the back stack and the user presses the *Back* button, the FragmentManager (/reference/android/app/FragmentManager.html) pops the most recent transaction off the back stack and performs the reverse action (such as removing a fragment if the transaction added it).

> **Note:** You **should not add transactions to the back stack** when the transaction is for horizontal navigation (such as when switching tabs) or when modifying the content appearance (such as when adjusting filters). For more information, about when *Back* navigation is appropriate, see the Navigation (/design/patterns/navigation.html) design guide.

If your application updates other user interface elements to reflect the current state of your fragments, such as the action bar, remember to update the UI when you commit the transaction. You should update your user interface after the back stack changes in addition to when you commit the transaction. You can listen for when a FragmentTransaction (/reference/android/app/FragmentTransaction.html) is reverted by setting up an FragmentManager.OnBackStackChangedListener (/reference/android/app/FragmentManager.OnBackStackChangedListener.html):

```
getSupportFragmentManager().addOnBackStackChangedListener(
        new FragmentManager.OnBackStackChangedListener() {
            public void onBackStackChanged() {
                // Update your UI here.
            }
        });
```

Implement Back Navigation for WebViews

If a part of your application is contained in a WebView (/reference/android/webkit/WebView.html), it may be appropriate for *Back* to traverse browser history. To do so, you can override onBackPressed() (/reference/android/app/Activity.html#onBackPressed()) and proxy to the WebView (/reference/android/webkit/WebView.html) if it has history state:

```
@Override
public void onBackPressed() {
    if (mWebView.canGoBack()) {
        mWebView.goBack();
        return;
    }
```

```
    // Otherwise defer to system default behavior.
    super.onBackPressed();
}
```

Be careful when using this mechanism with highly dynamic web pages that can grow a large history. Pages that generate an extensive history, such as those that make frequent changes to the document hash, may make it tedious for users to get out of your activity.

For more information about using `WebView` (/reference/android/webkit/WebView.html), read Building Web Apps in WebView (/guide/webapps/webview.html).