Providing Up Navigation

All screens in your app that are not the main entrance to your app (the "home" screen) should offer the user a way to navigate to the logical parent screen in the app's hierarchy by pressing the *Up* button in the action bar (/guide/topics/ui/actionbar.html). This lesson shows you how to properly implement this behavior.

THIS LESSON TEACHES YOU TO:

1. Specify the Parent Activity
2. Add Up Action
3. Navigate Up to Parent Activity

YOU SHOULD ALSO READ

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

TRY IT OUT

Download the sample app

EffectiveNavigation.zip

Up Navigation Design

The concepts and principles for *Up* navigation are described in Designing Effective Navigation (/training/design-navigation/ancestral-temporal.html) and the Navigation (/design/patterns/navigation.html) design guide.

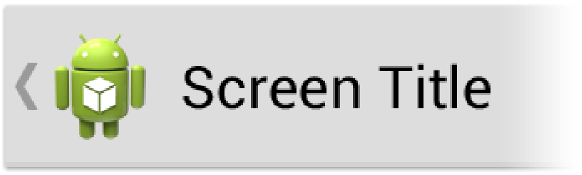

Figure 1. The *Up* button in the action bar.

Specify the Parent Activity

To implement *Up* navigation, the first step is to declare which activity is the appropriate parent for each activity. Doing so allows the system to facilitate navigation patterns such as *Up* because the system can determine the logical parent activity from the manifest file.

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.

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" >
        <!-- Parent activity meta-data to support 4.0 and lower -->
        <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 navigate *Up* to the appropriate parent using the NavUtils (/reference/android/support/v4/app/NavUtils.html) APIs, as shown in the following sections.

Add Up Action

To allow *Up* navigation with the app icon in the action bar, call setDisplayHomeAsUpEnabled() (/reference/android/app/ActionBar.html#setDisplayHomeAsUpEnabled(boolean)):

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    getActionBar().setDisplayHomeAsUpEnabled(true);
}
```

This adds a left-facing caret alongside the app icon and enables it as an action button such that when the user presses it, your activity receives a call to onOptionsItemSelected() (/reference/android/app/Activity.html#onOptionsItemSelected(android.view.MenuItem)). The ID for the action is `android.R.id.home`.

Navigate Up to Parent Activity

To navigate up when the user presses the app icon, you can use the NavUtils (/reference/android/support/v4/app/NavUtils.html) class's static method, navigateUpFromSameTask() (/reference/android/support/v4/app/NavUtils.html#navigateUpFromSameTask(android.app.Activity)). When you call this method, it finishes the current activity and starts (or resumes) the appropriate parent activity. If the target parent activity is in the task's back stack, it is brought forward as defined by FLAG_ACTIVITY_CLEAR_TOP (/reference/android/content/Intent.html#FLAG_ACTIVITY_CLEAR_TOP).

For example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    // Respond to the action bar's Up/Home button
    case android.R.id.home:
        NavUtils.navigateUpFromSameTask(this);
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

However, using navigateUpFromSameTask() (/reference/android/support/v4/app/NavUtils.html#navigateUpFromSameTask(android.app.Activity)) is suitable **only when your app is the owner of the current task** (that is, the user began this task from your app). If that's not true and your activity was started in a task that belongs to a different app, then navigating *Up* should create a new task that belongs to your app, which requires that you create a new back stack.

Navigate up with a new back stack

If your activity provides any intent filters (/guide/components/intents-filters.html#ifs) that allow other apps to start the activity, you should implement the onOptionsItemSelected() (/reference/android/app/Activity.html#onOptionsItemSelected(android.view.MenuItem)) callback such that if the user presses the *Up* button after entering your activity from another app's task, your app starts a new task with the appropriate back stack before navigating up.

You can do so by first calling `shouldUpRecreateTask()` (/reference/android/support/v4/app/NavUtils.html#shouldUpRecreateTask(android.app.Activity, android.content.Intent)) to check whether the current activity instance exists in a different app's task. If it returns true, then build a new task with `TaskStackBuilder` (/reference/android/support/v4/app/TaskStackBuilder.html). Otherwise, you can use the `navigateUpFromSameTask()` (/reference/android/support/v4/app/NavUtils.html#navigateUpFromSameTask(android.app.Activity)) method as shown above.

For example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    // Respond to the action bar's Up/Home button
    case android.R.id.home:
        Intent upIntent = NavUtils.getParentActivityIntent(this);
        if (NavUtils.shouldUpRecreateTask(this, upIntent)) {
            // This activity is NOT part of this app's task, so create a new task
            // when navigating up, with a synthesized back stack.
            TaskStackBuilder.create(this)
                    // Add all of this activity's parents to the back stack
                    .addNextIntentWithParentStack(upIntent)
                    // Navigate up to the closest parent
                    .startActivities();
        } else {
            // This activity is part of this app's task, so simply
            // navigate up to the logical parent activity.
            NavUtils.navigateUpTo(this, upIntent);
        }
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

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.