Using Intent Filters for Plug-ins and Extensibility

So far you've learned how to explicitly create implicit Intents, but that's only half the story. Android lets future packages provide new functionality for existing applications, using Intent Filters to populate menus dynamically at run time.

This provides a plug-in model for your Activities that lets them take advantage of functionality you haven't yet conceived of through new application components, without your having to modify or recompile your projects.

The addIntentOptions method available from the Menu class lets you specify an Intent that describes the data that is acted upon by this menu. Android resolves this Intent and returns every action specified in the Intent Filters that matches the specifi ed data. A new Menu Item is created for each, with the text populated from the matching Intent Filters' labels.

The elegance of this concept is best explained by example. Say the data your application displays are a list of places. At the moment, the menu actions available might include "view" and "show directions to." Jump a few years ahead, and you've created an application that interfaces with your car, allowing your phone to handle driving. Thanks to the runtime menu generation, by including a new Intent Filter — with a DRIVE_CAR action — within the new Activity's node, Android will automagically add this action as a new Menu Item in your earlier application.

Runtime menu population provides the ability to retrofi t functionality when you create new components capable of performing actions on a given type of data. Many of Android's native applications use this functionality, giving you the ability to provide additional actions to native Activities.

Supplying Anonymous Actions to Applications

To make actions available for other Activities, publish them using intent-filter tags within their manifest nodes.

The Intent Filter describes the action it performs and the data upon which it can be performed. The latter will be used during the Intent resolution process to determine when this action should be available.

The category tag must be either or both ALTERNATIVE and SELECTED_ALTERNATIVE. The text used by Menu Items is specifi ed by the android:label attribute.

The following XML shows an example of an Intent Filter used to advertise an Activity's ability to nuke moon bases from orbit.

```
<activity android:name=".NostromoController">
<intent-filter android:label="Nuke From Orbit">
<action android:name="com.pad.nostromo.NUKE_FROM_ORBIT"/>
<data android:mimeType="vnd.moonbase.cursor.item/*"/>
<category android:name="android.intent.category.ALTERNATIVE"/>
<category
android:name="android.intent.category.SELECTED_ALTERNATIVE"
/>
</intent-filter>
</activity>
```

The Content Provider and other code needed for this example to run aren't provided; in the following sections, you'll see how to write the code that will make this action available dynamically from another Activity's menu.

Incorporating Anonymous Actions in Your Activity's Menu

To add menu options to your menus at run time, you use the addIntentOptions method on the menu object in question, passing in an Intent that specifi es the data for which you want to provide actions. Generally, this will be handled within your Activity's onCreateOptionsMenu or onCreateContextMenu handlers.

The Intent you create will be used to resolve components with Intent Filters that supply actions for the data you specify. The Intent is being used to fi nd actions, so don't assign it one; it should only specify the data on which to perform actions. You should also specify the category of the action, either CATEGORY_ALTERNATIVE or CATEGORY_SELECTED_ALTERNATIVE.

The skeleton code for creating an Intent for menu-action resolution is shown below:

```
Intent intent = new Intent();
intent.setData(MyProvider.CONTENT_URI);
intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
```

Pass this Intent into addIntentOptions on the menu you wish to populate, as well as any option fl ags, the name of the calling class, the menu group to use, and menu ID values. You can also specify an array of Intents you'd like to use to create additional Menu Items.

The following code snippet gives an idea of how to dynamically populate an Activity menu that would include the "moonbase nuker" action from the previous section:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
super.onCreateOptionsMenu(menu);
// Create the intent used to resolve which actions
// should appear in the menu.
Intent intent = new Intent();
intent.setData(MoonBaseProvider.CONTENT_URI);
intent.addCategory(Intent.CATEGORY_SELECTED_ALTERNATIVE);
// Normal menu options to let you set a group and ID
// values for the menu items you're adding.
int menuGroup = 0;
int menuItemId = 0;
int menuItemOrder = Menu.NONE;
// Provide the name of the component that's calling
// the action -- generally the current Activity.
ComponentName caller = getComponentName();
// Define intents that should be added first.
Intent[] specificIntents = null;
// The menu items created from the previous Intents
// will populate this array.
MenuItem[] outSpecificItems = null;
// Set any optional flags.
int flags = Menu.FLAG_APPEND_TO_GROUP;
// Populate the menu
menu.addIntentOptions(menuGroup,
menuItemId,
menuItemOrder,
caller,
specificIntents,
intent,
flags,
outSpecificItems);
return true;
}
```