Creating a Search Interface

In this document

Key classes

- `SearchManager`
- `SearchView`

Related samples

- Searchable Dictionary
- SearchView in the Action Bar
- SearchView filter mode

Downloads

- Action Bar Icon Pack

When you're ready to add search functionality to your application, Android helps you implement the user interface with either a search dialog that appears at the top of the activity window or a search widget that you can insert in your layout. Both the search dialog and the widget can deliver the user's search query to a specific activity in your application. This way, the user can initiate a search from any activity where the search dialog or widget is available, and the system starts the appropriate activity to perform the search and present results.

Other features available for the search dialog and widget include:

- Voice search
- Search suggestions based on recent queries
- Search suggestions that match actual results in your application data

This guide shows you how to set up your application to provide a search interface that's assisted by the Android system to deliver search queries, using either the search dialog or the search widget.

The Basics

The Basics

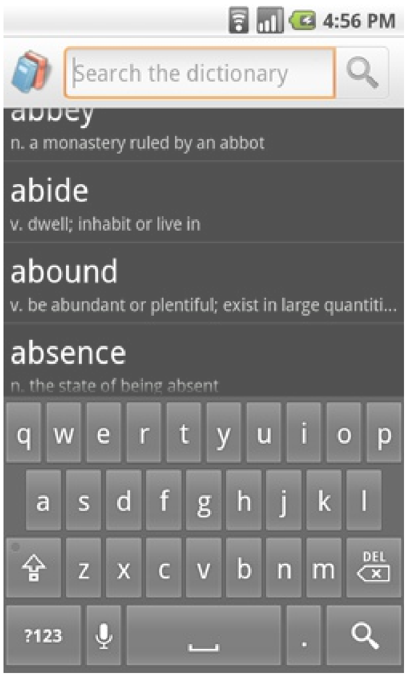
4:56 PM
Search the dictionary
abbey
n. a monastery ruled by an abbot
abide
v. dwell; inhabit or live in
abound
v. be abundant or plentiful; exist in large quantiti...
absence
n. the state of being absent

Figure 1. Screenshot of an application's search dialog.

Before you begin, you should decide whether you'll implement your search interface using the search dialog or the search widget. Both provide the same search features, but in slightly different ways:

- The **search dialog** is a UI component that's controlled by the Android system. When activated by the user, the search dialog appears at the top of the activity, as shown in figure 1.

  The Android system controls all events in the search dialog. When the user submits a query, the system delivers the query to the activity that you specify to handle searches. The dialog can also provide search suggestions while the user types.

- The **search widget** is an instance of `SearchView` that you can place anywhere in your layout. By default, the search widget behaves like a standard `EditText` widget and doesn't do anything, but you can configure it so that the Android system handles all input events, delivers queries to the appropriate activity, and provides search suggestions (just like the search dialog).

> **Note:** If you want, you can handle all user input into the search widget yourself, using various callback methods and listeners. This document, however, focuses on how to integrate the search widget with the system for an assisted search implementation. If you want to handle all user input yourself, read the reference documentation for `SearchView` and its nested interfaces.

When the user executes a search from the search dialog or a search widget, the system creates an `Intent` and stores the user query in it. The system then starts the activity that you've declared to handle searches (the "searchable activity") and delivers it the intent. To set up your application for this kind of assisted search, you need the following:

- A searchable configuration

  An XML file that configures some settings for the search dialog or widget. It includes settings for features such as voice search, search suggestion, and hint text for the search box.

- A searchable activity

  The `Activity` that receives the search query, searches your data, and displays the search results.

- A search interface, provided by either:
  - The search dialog

    By default, the search dialog is hidden, but appears at the top of the screen when you call `onSearchRequested()` (when the user presses your Search button).

  - Or, a `SearchView` widget

    Using the search widget allows you to put the search box anywhere in your activity. Instead of putting it in your activity layout, you should usually use SearchView as an action view in the app bar.

The rest of this document shows you how to create the searchable configuration, searchable activity, and implement a search interface with either the search dialog or search widget.

Creating a Searchable Configuration

The first thing you need is an XML file called the searchable configuration. It configures certain UI aspects of the search dialog or widget and defines how features such as suggestions and voice search behave. This file is traditionally named `searchable.xml` and must be saved in the `res/xml/` project directory.

> **Note:** The system uses this file to instantiate a `SearchableInfo` object, but you cannot create this object yourself at runtime—you must declare the searchable configuration in XML.

The searchable configuration file must include the `<searchable>` element as the root node and specify one or more attributes. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_label"
    android:hint="@string/search_hint" >
</searchable>
```

The `android:label` attribute is the only required attribute. It points to a string resource, which should be the application name. This label isn't actually visible to the user until you enable search suggestions for Quick Search Box. At that point, this label is visible in the list of Searchable items in the system Settings.

Though it's not required, we recommend that you always include the `android:hint` attribute, which provides a hint string in the search box before users enters a query. The hint is important because it provides important clues to users about what they can search.

> **Tip:** For consistency among other Android applications, you should format the string for `android:hint` as "Search <content-or-product>". For example, "Search songs and artists" or "Search YouTube".

The `<searchable>` element accepts several other attributes. However, you don't need most attributes until you add features such as search suggestions and voice search. For detailed information about the searchable configuration file, see the Searchable Configuration reference document.

Creating a Searchable Activity

A searchable activity is the `Activity` in your application that performs searches based on a query string and presents the search results.

When the user executes a search in the search dialog or widget, the system starts your searchable activity and delivers it the search query in an `Intent` with the `ACTION_SEARCH` action. Your searchable activity retrieves the query from the intent's `QUERY` extra, then searches your data and presents the results.

Because you may include the search dialog or widget in any other activity in your application, the system must know which activity is your searchable activity, so it can properly deliver the search query. So, you must first declare your searchable activity in the Android manifest file.

Declaring a searchable activity

If you don't have one already, create an `Activity` that will perform searches and present results. You don't need to implement the search functionality yet—just create an activity that you can declare in the manifest. Inside the manifest's `<activity>` element:

1. Declare the activity to accept the `ACTION_SEARCH` intent, in an `<intent-filter>` element.

2. Specify the searchable configuration to use, in a `<meta-data>` element.

For example:

```
<application ... >
    <activity android:name=".SearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
                   android:resource="@xml/searchable"/>
    </activity>
    ...
</application>
```

The `<meta-data>` element must include the `android:name` attribute with a value of `"android.app.searchable"` and the `android:resource` attribute with a reference to the searchable configuration file (in this example, it refers to the `res/xml/searchable.xml` file).

> **Note:** The `<intent-filter>` does not need a `<category>` with the `DEFAULT` value (which you usually see in `<activity>` elements), because the system delivers the `ACTION_SEARCH` intent explicitly to your searchable activity, using its component name.

Performing a search

Once you have declared your searchable activity in the manifest, performing a search in your searchable activity involves three steps:

1. Receiving the query
2. Searching your data
3. Presenting the results

Traditionally, your search results should be presented in a `ListView`, so you might want your searchable activity to extend `ListActivity`. It includes a default layout with a single `ListView` and provides several convenience methods for working with the `ListView`.

Receiving the query

When a user executes a search from the search dialog or widget, the system starts your searchable activity and sends it a `ACTION_SEARCH` intent. This intent carries the search query in the `QUERY` string extra. You must check for this intent when the activity starts and extract the string. For example, here's how you can get the search query when your searchable activity starts:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.search);

    // Get the intent, verify the action and get the query
    Intent intent = getIntent();
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      String query = intent.getStringExtra(SearchManager.QUERY);
      doMySearch(query);
    }
}
```

The `QUERY` string is always included with the `ACTION_SEARCH` intent. In this example, the query is retrieved and passed to a local `doMySearch()` method where the actual search operation is done.

Searching your data

The process of storing and searching your data is unique to your application. You can store and search your data in many ways, but this guide does not show you how to store your data and search it. Storing and searching your data is something you should carefully consider in terms of your needs and your data format. However, here are some tips you might be able to apply:

- If your data is stored in a SQLite database on the device, performing a full-text search (using FTS3, rather than a `LIKE` query) can provide a more robust search across text data and can produce results significantly faster. See sqlite.org for information about FTS3 and the `SQLiteDatabase` class for information about SQLite on Android. Also look at the Searchable Dictionary sample application to see a complete SQLite implementation that performs searches with FTS3.

- If your data is stored online, then the perceived search performance might be inhibited by the user's data connection. You might want to display a spinning progress wheel until your search returns. See `android.net` for a reference of network APIs and Creating a Progress Dialog for information about how to display a progress wheel.

Regardless of where your data lives and how you search it, we recommend that you return search results to your searchable activity with an `Adapter`. This way, you can easily present all the search results in a `ListView`. If your data comes from a SQLite database query, you can apply your results to a `ListView` using a `CursorAdapter`. If your data comes in some other type of format, then you can create an extension of `BaseAdapter`.

About Adapters

An `Adapter` binds each item from a set of data into a `View` object. When the `Adapter` is applied to a `ListView`, each piece of data is inserted as an individual view into the list. `Adapter` is just an interface, so implementations such as `CursorAdapter` (for binding data from a `Cursor`) are needed. If none of the existing implementations work for your data, then you can implement your own from `BaseAdapter`. Install the SDK Samples package for API Level 4 to see the original version of the Searchable Dictionary, which creates a custom adapter to read data from a file.

Presenting the results

As discussed above, the recommended UI for your search results is a `ListView`, so you might want your searchable activity to extend `ListActivity`. You can then call `setListAdapter()`, passing it an `Adapter` that is bound to your data. This injects all the search results into the activity `ListView`.

For more help presenting your results in a list, see the `ListActivity` documentation.

Also see the Searchable Dictionary sample for an a complete demonstration of how to search an SQLite database and use an `Adapter` to provide results in a `ListView`.

Using the Search Dialog

The search dialog provides a floating search box at the top of the screen, with the application icon on the left. The search dialog can provide search suggestions as the user types and, when the user executes a search, the system sends the search query to a searchable activity that performs the search. However, if you are developing your application for devices running Android 3.0, you should consider using the search widget instead (see Using the Search Widget section).

The search dialog is always hidden by default, until the user activates it. Your application can activate the search dialog by calling `onSearchRequested()`. However, this method doesn't work until you enable the search dialog for the activity.

To enable the search dialog, you must indicate to the system which searchable activity should receive search queries from the search dialog, in order to perform searches. For example, in the previous section about Creating a Searchable Activity, a searchable activity named `SearchableActivity` was created. If you want a separate activity, named `OtherActivity`, to show the search dialog and deliver searches to `SearchableActivity`, you must declare in the manifest that `SearchableActivity` is the searchable activity to use for the search dialog in `OtherActivity`.

To declare the searchable activity for an activity's search dialog, add a `<meta-data>` element inside the respective activity's `<activity>` element. The `<meta-data>` element must include the `android:value` attribute that specifies the searchable activity's class name and the `android:name` attribute with a value of `"android.app.default_searchable"`.

For example, here is the declaration for both a searchable activity, `SearchableActivity`, and another activity, `OtherActivity`, which uses `SearchableActivity` to perform searches executed from its search dialog:

```
<application ... >
    <!-- this is the searchable activity; it performs searches -->
    <activity android:name=".SearchableActivity" >
        <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
        </intent-filter>
        <meta-data android:name="android.app.searchable"
                   android:resource="@xml/searchable"/>
    </activity>

    <!-- this activity enables the search dialog to initiate searches
         in the SearchableActivity -->
    <activity android:name=".OtherActivity" ... >
        <!-- enable the search dialog to send searches to SearchableActivity -->
        <meta-data android:name="android.app.default_searchable"
                   android:value=".SearchableActivity" />
    </activity>
    ...
</application>
```

Because the `OtherActivity` now includes a `<meta-data>` element to declare which searchable activity to use for searches, the activity has enabled the search dialog. While the user is in this activity, the `onSearchRequested()` method activates the search dialog. When the user executes the search, the system starts `SearchableActivity` and delivers it the `ACTION_SEARCH` intent.

> **Note:** The searchable activity itself provides the search dialog by default, so you don't need to add this declaration to `SearchableActivity`.

If you want every activity in your application to provide the search dialog, insert the above `<meta-data>` element as a child of the `<application>` element, instead of each `<activity>`. This way, every activity inherits the value, provides the search dialog, and delivers searches to the same searchable activity. (If you have multiple searchable activities, you can override the default searchable activity by placing a different `<meta-data>` declaration inside individual activities.)

With the search dialog now enabled for your activities, your application is ready to perform searches.

Invoking the search dialog

Although some devices provide a dedicated Search button, the behavior of the button may vary between devices and many devices do not provide a Search button at all. So when using the search dialog, you **must provide a search button in your UI** that activates the search dialog by calling `onSearchRequested()`.

For instance, you should add a Search button in your Options Menu or UI layout that calls `onSearchRequested()`. For consistency with the Android system and other apps, you should label your button with the Android Search icon that's available from the Action Bar Icon Pack.

> **Note:** If your app uses an app bar, then you should not use the search dialog for your search interface. Instead, use the search widget as a collapsible view in the app bar.

You can also enable "type-to-search" functionality, which activates the search dialog when the user starts typing on the keyboard—the keystrokes are inserted into the search dialog. You can enable type-to-search in your activity by calling `setDefaultKeyMode(DEFAULT_KEYS_SEARCH_LOCAL)` during your activity's `onCreate()` method.

The impact of the search dialog on your activity lifecycle

The search dialog is a `Dialog` that floats at the top of the screen. It does not cause any change in the activity stack, so when the search dialog appears, no lifecycle methods (such as `onPause()`) are called. Your activity just loses input focus, as input focus is given to the search dialog.

If you want to be notified when the search dialog is activated, override the `onSearchRequested()` method. When the system calls this method, it is an indication that your activity has lost input focus to the search dialog, so you can do any work appropriate for the event (such as pause a game). Unless you are passing search context data (discussed below), you should end the method by calling the super class implementation. For example:

```
@Override
public boolean onSearchRequested() {
    pauseSomeStuff();
    return super.onSearchRequested();
}
```

If the user cancels search by pressing the *Back* button, the search dialog closes and the activity regains input focus. You can register to be notified when the search dialog is closed with `setOnDismissListener()` and/or `setOnCancelListener()`. You should need to register only the `OnDismissListener`, because it is called every time the search dialog closes. The `OnCancelListener` only pertains to events in which the user explicitly exited the search dialog, so it is not called when a search is executed (in which case, the search dialog naturally disappears).

If the current activity is not the searchable activity, then the normal activity lifecycle events are triggered once the user executes a search (the current activity receives `onPause()` and so forth, as described in the Activities document). If, however, the current activity is the searchable activity, then one of two things happens:

a. By default, the searchable activity receives the `ACTION_SEARCH` intent with a call to `onCreate()` and a new instance of the activity is brought to the top of the activity stack. There are now two instances of your searchable activity in the activity stack (so pressing the *Back* button goes back to the previous instance of the searchable activity, rather than exiting the searchable activity).

b. If you set `android:launchMode` to `"singleTop"`, then the searchable activity receives the `ACTION_SEARCH` intent with a call to `onNewIntent(Intent)`, passing the new `ACTION_SEARCH` intent here. For example, here's how you might handle this case, in which the searchable activity's launch mode is `"singleTop"`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.search);
    handleIntent(getIntent());
}

@Override
protected void onNewIntent(Intent intent) {
    setIntent(intent);
    handleIntent(intent);
}

private void handleIntent(Intent intent) {
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      String query = intent.getStringExtra(SearchManager.QUERY);
      doMySearch(query);
    }
}
```

Compared to the example code in the section about Performing a Search, all the code to handle the search intent is now in the `handleIntent()` method, so that both `onCreate()` and `onNewIntent()` can execute it.

When the system calls `onNewIntent(Intent)`, the activity has not been restarted, so the `getIntent()` method returns the same intent that was received with `onCreate()`. This is why you should call `setIntent(Intent)` inside `onNewIntent(Intent)` (so that the intent saved by the activity is updated in case you call `getIntent()` in the future).

The second scenario using `"singleTop"` launch mode is usually ideal, because chances are good that once a search is done, the user will perform additional searches and it's a bad experience if your application creates multiple instances of the searchable activity. So, we recommend that you set your searchable activity to `"singleTop"` launch mode in the application manifest. For example:

```
<activity android:name=".SearchableActivity"
          android:launchMode="singleTop" >
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    <meta-data android:name="android.app.searchable"
                      android:resource="@xml/searchable"/>
  </activity>
```

Passing search context data

In some cases, you can make necessary refinements to the search query inside the searchable activity, for every search made. However, if you want to refine your search criteria based on the activity from which the user is performing a search, you can provide additional data in the intent that the system sends to your searchable activity. You can pass the additional data in the `APP_DATA` `Bundle`, which is included in the `ACTION_SEARCH` intent.

To pass this kind of data to your searchable activity, override the `onSearchRequested()` method for the activity from which the user can perform a search, create a `Bundle` with the additional data, and call `startSearch()` to activate the search dialog. For example:

```
@Override
public boolean onSearchRequested() {
     Bundle appData = new Bundle();
     appData.putBoolean(SearchableActivity.JARGON, true);
     startSearch(null, false, appData, false);
     return true;
 }
```

Returning "true" indicates that you have successfully handled this callback event and called `startSearch()` to activate the search dialog. Once the user submits a query, it's delivered to your searchable activity along with the data you've added. You can extract the extra data from the `APP_DATA` `Bundle` to refine the search. For example:

```
Bundle appData = getIntent().getBundleExtra(SearchManager.APP_DATA);
if (appData != null) {
    boolean jargon = appData.getBoolean(SearchableActivity.JARGON);
}
```

Caution: Never call the `startSearch()` method from outside the `onSearchRequested()` callback method. To activate the search dialog in your activity, always call `onSearchRequested()`. Otherwise, `onSearchRequested()` is not called and customizations (such as the addition of `appData` in the above example) are missed.

Using the Search Widget

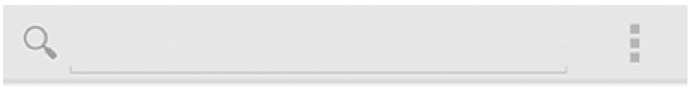

Figure 2. The `SearchView` widget as an "action view" in the Action Bar.

The `SearchView` widget is available in Android 3.0 and higher. If you're developing your application for Android 3.0 and have decided to use the search widget, we recommend that you insert the search widget as an action view in the app bar, instead of using the search dialog (and instead of placing the search widget in your activity layout). For example, figure 2 shows the search widget in the app bar.

The search widget provides the same functionality as the search dialog. It starts the appropriate activity when the user executes a search, and it can provide search suggestions and perform voice search. If it's not an option for you to put the search widget in the Action Bar, you can instead put the search widget somewhere in your activity layout.

Note: When you use the search widget as an action view, you still might need to support using the search dialog, for cases in which the search widget does not fit in the Action Bar. See the following section about Using both the widget and the dialog.

Configuring the search widget

After you've created a searchable configuration and a searchable activity, as discussed above, you need to enable assisted search for each `SearchView`. You can do so by calling `setSearchableInfo()` and passing it the `SearchableInfo` object that represents your searchable configuration.

You can get a reference to the `SearchableInfo` by calling `getSearchableInfo()` on `SearchManager`.

For example, if you're using a `SearchView` as an action view in the app bar, you should enable the widget during the `onCreateOptionsMenu()` callback:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the options menu from XML
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager = (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView = (SearchView) menu.findItem(R.id.menu_search).getActionView();
    // Assumes current activity is the searchable activity
    searchView.setSearchableInfo(searchManager.getSearchableInfo(getComponentName()));
    searchView.setIconifiedByDefault(false); // Do not iconify the widget; expand it by default

    return true;
}
```

That's all you need. The search widget is now configured and the system will deliver search queries to your searchable activity. You can also enable search suggestions for the search widget.

> **Note:** If you want to handle all user input yourself, you can do so with some callback methods and event listeners. For more information, see the reference documentation for `SearchView` and its nested interfaces for the appropriate event listeners.

For more information about action views in the Action Bar, see Action Views and Action Providers.

Other search widget features

The `SearchView` widget allows for a few additional features you might want:

A submit button

By default, there's no button to submit a search query, so the user must press the "Return" key on the keyboard to initiate a search. You can add a "submit" button by calling `setSubmitButtonEnabled(true)`.

Query refinement for search suggestions

When you've enabled search suggestions, you usually expect users to simply select a suggestion, but they might also want to refine the suggested search query. You can add a button alongside each suggestion that inserts the suggestion in the search box for refinement by the user, by calling `setQueryRefinementEnabled(true)`.

The ability to toggle the search box visibility

By default, the search widget is "iconified," meaning that it is represented only by a search icon (a magnifying glass), and expands to show the search box when the user touches it. As shown above, you can show the search box by default, by calling `setIconifiedByDefault(false)`. You can also toggle the search widget appearance by calling `setIconified()`.

There are several other APIs in the `SearchView` class that allow you to customize the search widget. However, most of them are used only when you handle all user input yourself, instead of using the Android system to deliver search queries and display search suggestions.

Using both the widget and the dialog

If you insert the search widget in the Action Bar as an action view, and you enable it to appear in the Action Bar "if there is room" (by setting `android:showAsAction="ifRoom"`), then there is a chance that the search widget will not appear as an action view, but the menu item will appear in the overflow menu. For example, when your application runs on a smaller screen, there might not be enough room in the Action Bar to display the search widget along with other action items or navigation elements, so the menu item will instead appear in the overflow menu. When placed in the overflow menu, the item works like an ordinary menu item and does not display the action view (the search widget).

To handle this situation, the menu item to which you've attached the search widget should activate the search dialog when the user selects it from the overflow menu. In order for it to do so, you must implement `onOptionsItemSelected()` to handle the "Search" menu item and open the search dialog by calling `onSearchRequested()`.

For more information about how items in the Action Bar work and how to handle this situation, see the Action Bar developer guide.

Also see the Searchable Dictionary for an example implementation using both the dialog and the widget.

Adding Voice Search

You can add voice search functionality to your search dialog or widget by adding the `android:voiceSearchMode` attribute to your searchable configuration. This adds a voice search button that launches a voice prompt. When the user has finished speaking, the transcribed search query is sent to your searchable activity.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer" >
</searchable>
```

The value `showVoiceSearchButton` is required to enable voice search, while the second value, `launchRecognizer`, specifies that the voice search button should launch a recognizer that returns the transcribed text to the searchable activity.

You can provide additional attributes to specify the voice search behavior, such as the language to be expected and the maximum number of results to return. See the Searchable Configuration reference for more information about the available attributes.

> **Note:** Carefully consider whether voice search is appropriate for your application. All searches performed with the voice search button are immediately sent to your searchable activity without a chance for the user to review the transcribed query. Sufficiently test the voice recognition and ensure that it understands the types of queries that the user might submit inside your application.

Adding Search Suggestions

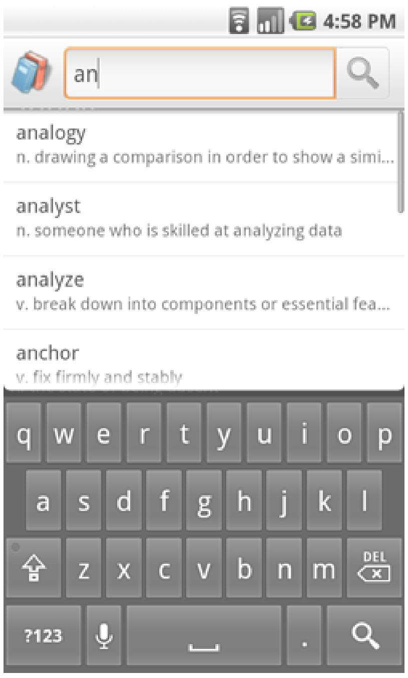
4:58 PM
an
analogy
n. drawing a comparison in order to show a simi...
analyst
n. someone who is skilled at analyzing data
analyze
v. break down into components or essential fea...
anchor
v. fix firmly and stably

Figure 3. Screenshot of a search dialog with custom search suggestions.

Both the search dialog and the search widget can provide search suggestions as the user types, with assistance from the Android system. The system manages the list of suggestions and handles the event when the user selects a suggestion.

You can provide two kinds of search suggestions:

Recent query search suggestions

These suggestions are simply words that the user previously used as search queries in your application.

See Adding Recent Query Suggestions.

Custom search suggestions

These are search suggestions that you provide from your own data source, to help users immediately select the correct spelling or item they are searching for. Figure 3 shows an example of custom suggestions for a dictionary application—the user can select a suggestion to instantly go to the definition.

See Adding Custom Suggestions