Adding Custom Suggestions

In this document

Key classes

- `SearchManager`
- `SearchRecentSuggestionsProvider`
- `ContentProvider`

Related samples

- Searchable Dictionary

See also

- Searchable Configuration
- Content Providers

When using the Android search dialog or search widget, you can provide custom search suggestions that are created from data in your application. For example, if your application is a word dictionary, you can suggest words from the dictionary that match the text entered so far. These are the most valuable suggestions, because you can effectively predict what the user wants and provide instant access to it. Figure 1 shows an example of a search dialog with custom suggestions.

Once you provide custom suggestions, you can also make them available to the system-wide Quick Search Box, providing access to your content from outside your application.

Before you begin with this guide to add custom suggestions, you need to have implemented the Android search dialog or a search widget for searches in your application. If you haven't, see Creating a Search Interface.

The Basics

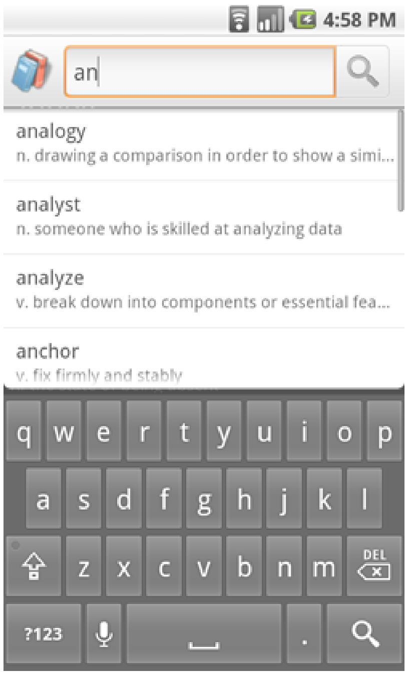
4:58 PM
an
analogy
n. drawing a comparison in order to show a simi...
analyst
n. someone who is skilled at analyzing data
analyze
v. break down into components or essential fea...
anchor
v. fix firmly and stably

Figure 1. Screenshot of a search dialog with custom search suggestions.

When the user selects a custom suggestion, the Android system sends an `Intent` to your searchable activity. Whereas a normal search query sends an intent with the `ACTION_SEARCH` action, you can instead define your custom suggestions to use `ACTION_VIEW` (or any other intent action), and also include data that's relevant to the selected suggestion. Continuing the dictionary example, when the user selects a suggestion, your application can immediately open the definition for that word, instead of searching the dictionary for matches.

To provide custom suggestions, do the following:

- Implement a basic searchable activity, as described in Creating a Search Interface.
- Modify the searchable configuration with information about the content provider that provides custom suggestions.
- Build a table (such as in an `SQLiteDatabase`) for your suggestions and format the table with required columns.
- Create a Content Provider that has access to your suggestions table and declare the provider in your manifest.
- Declare the type of `Intent` to be sent when the user selects a suggestion (including a custom action and custom data).

Just as the Android system displays the search dialog, it also displays your search suggestions. All you need is a content provider from which the system can retrieve your suggestions. If you're not familiar with creating content providers, read the Content Providers developer guide before you continue.

When the system identifies that your activity is searchable and provides search suggestions, the following procedure takes place when the user types a query:

1. The system takes the search query text (whatever has been typed so far) and performs a query to your content provider that manages your suggestions.
2. Your content provider returns a `Cursor` that points to all suggestions that are relevant to the search query text.
3. The system displays the list of suggestions provided by the Cursor.

Once the custom suggestions are displayed, the following might happen:

- If the user types another key, or changes the query in any way, the above steps are repeated and the suggestion list is updated as appropriate.
- If the user executes the search, the suggestions are ignored and the search is delivered to your searchable activity using the normal `ACTION_SEARCH` intent.
- If the user selects a suggestion, an intent is sent to your searchable activity, carrying a custom action and custom data so that your application can open the suggested content.

Modifying the searchable configuration

To add support for custom suggestions, add the `android:searchSuggestAuthority` attribute to the `<searchable>` element in your searchable configuration file. 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"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider">
</searchable>
```

You might need some additional attributes, depending on the type of intent you attach to each suggestion and how you want to format queries to your content provider. The other optional attributes are discussed in the following sections.

Creating a Content Provider

Creating a content provider for custom suggestions requires previous knowledge about content providers that's covered in the Content Provider developer guide. For the most part, a content provider for custom suggestions is the same as any other content provider. However, for each suggestion you provide, the respective row in the `Cursor` must include specific columns that the system understands and uses to format the suggestions.

When the user starts typing into the search dialog or search widget, the system queries your content provider for suggestions by calling `query()` each time a letter is typed. In your implementation of `query()`, your content provider must search your suggestion data and return a `Cursor` that points to the rows you have determined to be good suggestions.

Details about creating a content provider for custom suggestions are discussed in the following two sections:

Handling the suggestion query

How the system sends requests to your content provider and how to handle them

Building a suggestion table

How to define the columns that the system expects in the `Cursor` returned with each query

Handling the suggestion query

When the system requests suggestions from your content provider, it calls your content provider's `query()` method. You must implement this method to search your suggestion data and return a `Cursor` pointing to the suggestions you deem relevant.

Here's a summary of the parameters that the system passes to your `query()` method (listed in order):

`uri`

Always a content `Uri`, formatted as:

```
content://your.authority/optional.suggest.path/SUGGEST_URI_PATH_QUERY
```

The default behavior is for system to pass this URI and append it with the query text. For example:

```
content://your.authority/optional.suggest.path/SUGGEST_URI_PATH_QUERY/puppies
```

The query text on the end is encoded using URI encoding rules, so you might need to decode it before performing a search.

The `optional.suggest.path` portion is only included in the URI if you have set such a path in your searchable configuration file with the `android:searchSuggestPath` attribute. This is only needed if you use the same content provider for multiple searchable activities, in which case, you need to disambiguate the source of the suggestion query.

> **Note:** SUGGEST_URI_PATH_QUERY is not the literal string provided in the URI, but a constant that you should use if you need to refer to this path.

projection

Always null

selection

The value provided in the android:searchSuggestSelection attribute of your searchable configuration file, or null if you have not declared the android:searchSuggestSelection attribute. More about using this to get the query below.

selectionArgs

Contains the search query as the first (and only) element of the array if you have declared the android:searchSuggestSelection attribute in your searchable configuration. If you have not declared android:searchSuggestSelection, then this parameter is null. More about using this to get the query below.

sortOrder

Always null

The system can send you the search query text in two ways. The default manner is for the query text to be included as the last path of the content URI passed in the uri parameter. However, if you include a selection value in your searchable configuration's android:searchSuggestSelection attribute, then the query text is instead passed as the first element of the selectionArgs string array. Both options are summarized next.

Get the query in the Uri

By default, the query is appended as the last segment of the uri parameter (a Uri object). To retrieve the query text in this case, simply use getLastPathSegment(). For example:

```
String query = uri.getLastPathSegment().toLowerCase();
```

This returns the last segment of the Uri, which is the query text entered by the user.

Get the query in the selection arguments

Instead of using the URI, you might decide it makes more sense for your query() method to receive everything it needs to perform the look-up and you want the selection and selectionArgs parameters to carry the appropriate values. In such a case, add the android:searchSuggestSelection attribute to your searchable configuration with your SQLite selection string. In the selection string, include a question mark ("?") as a placeholder for the actual search query. The system calls query() with the selection string as the selection parameter and the search query as the first element in the selectionArgs array.

For example, here's how you might form the android:searchSuggestSelection attribute to create a full-text search statement:

```
<?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"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:searchSuggestSelection="word MATCH ?">
</searchable>
```

With this configuration, your query() method delivers the selection parameter as "word MATCH ?" and the selectionArgs parameter as the search query. When you pass these to an SQLite query() method, as their respective arguments, they are synthesized together (the question mark is replaced with the query text). If you chose to receive suggestion queries this way and need to add wildcards to the query text, append (and/or prefix) them to the selectionArgs parameter, because this value is wrapped in quotes and inserted in place of the question mark.

Another new attribute in the example above is `android:searchSuggestIntentAction`, which defines the intent action sent with each intent when the user selects a suggestion. It is discussed further in the section about Declaring an Intent for Suggestions.

> **Tip:** If you don't want to define a selection clause in the `android:searchSuggestSelection` attribute, but would still like to receive the query text in the `selectionArgs` parameter, simply provide a non-null value for the `android:searchSuggestSelection` attribute. This triggers the query to be passed in `selectionArgs` and you can ignore the `selection` parameter. In this way, you can instead define the actual selection clause at a lower level so that your content provider doesn't have to handle it.

Building a suggestion table

When you return suggestions to the system with a `Cursor`, the system expects specific columns in each row. So, regardless of whether you decide to store your suggestion data in an SQLite database on the device, a database on a web server, or another format on the device or web, you must format the suggestions as rows in a table and present them with a `Cursor`. The system understands several columns, but only two are required:

> **Creating a Cursor without a table**
>
> If your search suggestions are not stored in a table format (such as an SQLite table) using the columns required by the system, then you can search your suggestion data for matches and then format them into the necessary table on each request. To do so, create a `MatrixCursor` using the required column names and then add a row for each suggestion using `addRow(Object[])`. Return the final product from your Content Provider's `query()` method.

`_ID`

A unique integer row ID for each suggestion. The system requires this in order to present suggestions in a `ListView`.

`SUGGEST_COLUMN_TEXT_1`

The string that is presented as a suggestion.

The following columns are all optional (and most are discussed further in the following sections):

`SUGGEST_COLUMN_TEXT_2`

A string. If your Cursor includes this column, then all suggestions are provided in a two-line format. The string in this column is displayed as a second, smaller line of text below the primary suggestion text. It can be null or empty to indicate no secondary text.

`SUGGEST_COLUMN_ICON_1`

A drawable resource, content, or file URI string. If your Cursor includes this column, then all suggestions are provided in an icon-plus-text format with the drawable icon on the left side. This can be null or zero to indicate no icon in this row.

`SUGGEST_COLUMN_ICON_2`

A drawable resource, content, or file URI string. If your Cursor includes this column, then all suggestions are provided in an icon-plus-text format with the icon on the right side. This can be null or zero to indicate no icon in this row.

`SUGGEST_COLUMN_INTENT_ACTION`

An intent action string. If this column exists and contains a value at the given row, the action defined here is used when forming the suggestion's intent. If the element is not provided, the action is taken from the `android:searchSuggestIntentAction` field in your searchable configuration. If your action is the same for all suggestions, it is more efficient to specify the action using `android:searchSuggestIntentAction` and omit this column.

`SUGGEST_COLUMN_INTENT_DATA`

A data URI string. If this column exists and contains a value at the given row, this is the data that is used when forming the suggestion's intent. If the element is not provided, the data is taken from the `android:searchSuggestIntentData` field in your searchable configuration. If neither source is provided, the intent's data field is null. If your data is the same for all suggestions, or can be described using a constant part and a specific ID, it is more efficient to specify it using `android:searchSuggestIntentData` and omit this column.

`SUGGEST_COLUMN_INTENT_DATA_ID`

A URI path string. If this column exists and contains a value at the given row, then "/" and this value is appended to the data field in the intent. This should only be used if the data field specified by the `android:searchSuggestIntentData` attribute in the searchable configuration has already been set to an appropriate base string.

`SUGGEST_COLUMN_INTENT_EXTRA_DATA`

Arbitrary data. If this column exists and contains a value at a given row, this is the *extra* data used when forming the suggestion's intent. If not provided, the intent's extra data field is null. This column allows suggestions to provide additional data that is included as an extra in the intent's `EXTRA_DATA_KEY` key.

`SUGGEST_COLUMN_QUERY`

If this column exists and this element exists at the given row, this is the data that is used when forming the suggestion's query, included as an extra in the intent's `QUERY` key. Required if suggestion's action is `ACTION_SEARCH`, optional otherwise.

`SUGGEST_COLUMN_SHORTCUT_ID`

Only used when providing suggestions for Quick Search Box. This column indicates whether a search suggestion should be stored as a shortcut and whether it should be validated. Shortcuts are usually formed when the user clicks a suggestion from Quick Search Box. If missing, the result is stored as a shortcut and never refreshed. If set to `SUGGEST_NEVER_MAKE_SHORTCUT`, the result is not stored as a shortcut. Otherwise, the shortcut ID is used to check back for an up to date suggestion using `SUGGEST_URI_PATH_SHORTCUT`.

`SUGGEST_COLUMN_SPINNER_WHILE_REFRESHING`

Only used when providing suggestions for Quick Search Box. This column specifies that a spinner should be shown instead of an icon from `SUGGEST_COLUMN_ICON_2` while the shortcut of this suggestion is being refreshed in Quick Search Box.

Some of these columns are discussed more in the following sections.

Declaring an Intent for Suggestions

When the user selects a suggestion from the list that appears below the search dialog or widget, the system sends a custom `Intent` to your searchable activity. You must define the action and data for the intent.

Declaring the intent action

The most common intent action for a custom suggestion is `ACTION_VIEW`, which is appropriate when you want to open something, like the definition for a word, a person's contact information, or a web page. However, the intent action can be any other action and can even be different for each suggestion.

Depending on whether you want all suggestions to use the same intent action, you can define the action in two ways:

a. Use the `android:searchSuggestIntentAction` attribute of your searchable configuration file to define the action for all suggestions.

   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"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.Intent.action.VIEW" >
</searchable>
```

b. Use the `SUGGEST_COLUMN_INTENT_ACTION` column to define the action for individual suggestions.

   Add the `SUGGEST_COLUMN_INTENT_ACTION` column to your suggestions table and, for each suggestion, place in it the action to use (such as `"android.Intent.action.VIEW"`).

You can also combine these two techniques. For instance, you can include the `android:searchSuggestIntentAction` attribute with an action to be used with all suggestions by default, then override this action for some suggestions by declaring a different action in the

SUGGEST_COLUMN_INTENT_ACTION column. If you do not include a value in the SUGGEST_COLUMN_INTENT_ACTION column, then the intent provided in the android:searchSuggestIntentAction attribute is used.

> **Note**: If you do not include the android:searchSuggestIntentAction attribute in your searchable configuration, then you *must* include a value in the SUGGEST_COLUMN_INTENT_ACTION column for every suggestion, or the intent will fail.

Declaring intent data

When the user selects a suggestion, your searchable activity receives the intent with the action you've defined (as discussed in the previous section), but the intent must also carry data in order for your activity to identify which suggestion was selected. Specifically, the data should be something unique for each suggestion, such as the row ID for the suggestion in your SQLite table. When the intent is received, you can retrieve the attached data with getData() or getDataString().

You can define the data included with the intent in two ways:

a. Define the data for each suggestion inside the SUGGEST_COLUMN_INTENT_DATA column of your suggestions table.

   Provide all necessary data information for each intent in the suggestions table by including the SUGGEST_COLUMN_INTENT_DATA column and then populating it with unique data for each row. The data from this column is attached to the intent exactly as you define it in this column. You can then retrieve it with with getData() or getDataString().

   > **Tip**: It's usually easiest to use the table's row ID as the Intent data, because it's always unique. And the easiest way to do that is by using the SUGGEST_COLUMN_INTENT_DATA column name as an alias for the row ID column. See the Searchable Dictionary sample app for an example in which SQLiteQueryBuilder creates a projection map of column names to aliases.

b. Fragment a data URI into two pieces: the portion common to all suggestions and the portion unique to each suggestion. Place these parts into the android:searchSuggestintentData attribute of the searchable configuration and the SUGGEST_COLUMN_INTENT_DATA_ID column of your suggestions table, respectively.

   Declare the piece of the URI that is common to all suggestions in the android:searchSuggestIntentData attribute of your searchable configuration. 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"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:searchSuggestIntentData="content://com.example/datatable" >
</searchable>
```

   Then include the final path for each suggestion (the unique part) in the SUGGEST_COLUMN_INTENT_DATA_ID column of your suggestions table. When the user selects a suggestion, the system takes the string from android:searchSuggestIntentData, appends a slash ("/") and then adds the respective value from the SUGGEST_COLUMN_INTENT_DATA_ID column to form a complete content URI. You can then retrieve the Uri with with getData().

Add more data

If you need to express even more information with your intent, you can add another table column, SUGGEST_COLUMN_INTENT_EXTRA_DATA, which can store additional information about the suggestion. The data saved in this column is placed in EXTRA_DATA_KEY of the intent's extra Bundle.

Handling the Intent

Now that you provide custom search suggestions with custom intents, you need your searchable activity to handle these intents when the user selects a suggestion. This is in addition to handling the ACTION_SEARCH intent, which your searchable activity already does. Here's an example of how you can handle the intents during your activity onCreate() callback:

```
Intent intent = getIntent();
if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
    // Handle the normal search query case
    String query = intent.getStringExtra(SearchManager.QUERY);
    doSearch(query);
} else if (Intent.ACTION_VIEW.equals(intent.getAction())) {
    // Handle a suggestions click (because the suggestions all use ACTION_VIEW)
    Uri data = intent.getData();
    showResult(data);
}
```

In this example, the intent action is `ACTION_VIEW` and the data carries a complete URI pointing to the suggested item, as synthesized by the `android:searchSuggestIntentData` string and `SUGGEST_COLUMN_INTENT_DATA_ID` column. The URI is then passed to the local `showResult()` method that queries the content provider for the item specified by the URI.

> **Note:** You do *not* need to add an intent filter to your Android manifest file for the intent action you defined with the `android:searchSuggestIntentAction` attribute or `SUGGEST_COLUMN_INTENT_ACTION` column. The system opens your searchable activity by name to deliver the suggestion's intent, so the activity does not need to declare the accepted action.

Rewriting the query text

If the user navigates through the suggestions list using the directional controls (such as with a trackball or d-pad), the query text does not update, by default. However, you can temporarily rewrite the user's query text as it appears in the text box with a query that matches the suggestion currently in focus. This enables the user to see what query is being suggested (if appropriate) and then select the search box and edit the query before dispatching it as a search.

You can rewrite the query text in the following ways:

a. Add the `android:searchMode` attribute to your searchable configuration with the "queryRewriteFromText" value. In this case, the content from the suggestion's `SUGGEST_COLUMN_TEXT_1` column is used to rewrite the query text.

b. Add the `android:searchMode` attribute to your searchable configuration with the "queryRewriteFromData" value. In this case, the content from the suggestion's `SUGGEST_COLUMN_INTENT_DATA` column is used to rewrite the query text. This should only be used with URI's or other data formats that are intended to be user-visible, such as HTTP URLs. Internal URI schemes should not be used to rewrite the query in this way.

c. Provide a unique query text string in the `SUGGEST_COLUMN_QUERY` column of your suggestions table. If this column is present and contains a value for the current suggestion, it is used to rewrite the query text (and override either of the previous implementations).

Exposing search suggestions to Quick Search Box

Once you configure your application to provide custom search suggestions, making them available to the globally accessible Quick Search Box is as easy as modifying your searchable configuration to include `android:includeInGlobalSearch` as "true".

The only scenario in which additional work is necessary is when your content provider demands a read permission. In which case, you need to add a special `<path-permission>` element for the provider to grant Quick Search Box read access to your content provider. For example:

```
<provider android:name="MySuggestionProvider"
          android:authorities="com.example.MyCustomSuggestionProvider"
          android:readPermission="com.example.provider.READ_MY_DATA"
          android:writePermission="com.example.provider.WRITE_MY_DATA">
  <path-permission android:pathPrefix="/search_suggest_query"
                   android:readPermission="android.permission.GLOBAL_SEARCH" />
</provider>
```

In this example, the provider restricts read and write access to the content. The `<path-permission>` element amends the restriction by granting read access to content inside the `"/search_suggest_query"` path prefix when the `"android.permission.GLOBAL_SEARCH"` permission exists. This grants access to Quick Search Box so that it may query your content provider for suggestions.

If your content provider does not enforce read permissions, then Quick Search Box can read it by default.

Enabling suggestions on a device

When your application is configured to provide suggestions in Quick Search Box, it is not actually enabled to provide suggestions in Quick Search Box, by default. It is the user's choice whether to include suggestions from your application in the Quick Search Box. To enable search suggestions from your application, the user must open "Searchable items" (in Settings > Search) and enable your application as a searchable item.

Each application that is available to Quick Search Box has an entry in the Searchable items settings page. The entry includes the name of the application and a short description of what content can be searched from the application and made available for suggestions in Quick Search Box. To define the description text for your searchable application, add the `android:searchSettingsDescription` attribute to your searchable configuration. 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"
    android:searchSuggestAuthority="com.example.MyCustomSuggestionProvider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:includeInGlobalSearch="true"
    android:searchSettingsDescription="@string/search_description" >
</searchable>
```

The string for `android:searchSettingsDescription` should be as concise as possible and state the content that is searchable. For example, "Artists, albums, and tracks" for a music application, or "Saved notes" for a notepad application. Providing this description is important so the user knows what kind of suggestions are provided. You should always include this attribute when `android:includeInGlobalSearch` is "true".

Remember that the user must visit the settings menu to enable search suggestions for your application before your search suggestions appear in Quick Search Box. As such, if search is an important aspect of your application, then you might want to consider a way to convey that to your users — you might provide a note the first time they launch the app that instructs them how to enable search suggestions for Quick Search Box.

Managing Quick Search Box suggestion shortcuts

Suggestions that the user selects from Quick Search Box can be automatically made into shortcuts. These are suggestions that the system has copied from your content provider so it can quickly access the suggestion without the need to re-query your content provider.

By default, this is enabled for all suggestions retrieved by Quick Search Box, but if your suggestion data changes over time, then you can request that the shortcuts be refreshed. For instance, if your suggestions refer to dynamic data, such as a contact's presence status, then you should request that the suggestion shortcuts be refreshed when shown to the user. To do so, include the `SUGGEST_COLUMN_SHORTCUT_ID` in your suggestions table. Using this column, you can configure the shortcut behavior for each suggestion in one of the following ways:

a. Have Quick Search Box re-query your content provider for a fresh version of the suggestion shortcut.

   Provide a value in the `SUGGEST_COLUMN_SHORTCUT_ID` column and the suggestion is re-queried for a fresh version each time the shortcut is displayed. The shortcut is quickly displayed with whatever data was most recently available until the refresh query returns, at which point the suggestion is refreshed with the new information. The refresh query is sent to your content provider with a URI path of `SUGGEST_URI_PATH_SHORTCUT` (instead of `SUGGEST_URI_PATH_QUERY`).

   The `Cursor` you return should contain one suggestion using the same columns as the original suggestion, or be empty, indicating that the shortcut is no longer valid (in which case, the suggestion disappears and the shortcut is removed).

   If a suggestion refers to data that could take longer to refresh, such as a network-based refresh, you can also add the `SUGGEST_COLUMN_SPINNER_WHILE_REFRESHING` column to your suggestions table with a value of "true" in order to show a progress spinner for the right hand icon until the refresh is complete. Any value other than "true" does not show the progress spinner.

b. Prevent the suggestion from being copied into a shortcut at all.

   Provide a value of `SUGGEST_NEVER_MAKE_SHORTCUT` in the `SUGGEST_COLUMN_SHORTCUT_ID` column. In this case, the suggestion is never copied into a shortcut. This should only be necessary if you absolutely do not want the previously copied suggestion to appear. (Recall that if you provide a normal value for the column, then the suggestion shortcut appears only until the refresh query returns.)

c. Allow the default shortcut behavior to apply.

   Leave the `SUGGEST_COLUMN_SHORTCUT_ID` empty for each suggestion that will not change and can be saved as a shortcut.

If none of your suggestions ever change, then you do not need the `SUGGEST_COLUMN_SHORTCUT_ID` column at all.

> **Note**: Quick Search Box ultimately decides whether or not to create a shortcut for a suggestion, considering these values as a strong request from your application—there is no guarantee that the behavior you have requested for your suggestion shortcuts will be honored.

About Quick Search Box suggestion ranking

Once you make your application's search suggestions available to Quick Search Box, the Quick Search Box ranking determines how the suggestions are surfaced to the user for a particular query. This might depend on how many other apps have results for that query, and how often the user has selected your results compared to those from other apps. There is no guarantee about how your suggestions are ranked, or whether your app's suggestions show at all for a given query. In general, you can expect that providing quality results increases the likelihood that your app's suggestions are provided in a prominent position and apps that provide low quality suggestions are more likely to be ranked lower or not displayed.

See the Searchable Dictionary sample app for a complete demonstration of custom search suggestions.