**Loaders**

Introduced in Android 3.0, loaders make it easy to asynchronously（异步） load data in an activity or fragment. Loaders have these characteristics:

* They are available to every [Activity](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html) and [Fragment](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Fragment.html).
* They provide asynchronous loading of data.
* They monitor the source of their data and deliver new results when the content changes.
* They automatically reconnect to the last loader's cursor when being recreated after a configuration change. Thus, they don't need to re-query their data.

## Loader API Summary

There are multiple classes and interfaces that may be involved in using loaders in an application. They are summarized in this table:

|  |  |
| --- | --- |
| Class/Interface | Description |
| [LoaderManager](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\LoaderManager.html) | An abstract class associated with an [Activity](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html) or [Fragment](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Fragment.html) for managing one or more [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) instances. This helps an application manage longer-running operations in conjunction with the [Activity](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html)or [Fragment](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Fragment.html) lifecycle; the most common use of this is with a[CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html), however applications are free to write their own loaders for loading other types of data.   There is only one [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) per activity or fragment. But a[LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) can have multiple loaders. |
| [LoaderManager.LoaderCallbacks](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\LoaderManager.LoaderCallbacks.html) | A callback interface for a client to interact with the[LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html). For example, you use the [onCreateLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onCreateLoader(int, android.os.Bundle)) callback method to create a new loader. |
| [Loader](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Loader.html) | An abstract class that performs asynchronous loading of data. This is the base class for a loader. You would typically use[CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html), but you can implement your own subclass. While loaders are active they should monitor the source of their data and deliver new results when the contents change. |
| [AsyncTaskLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\AsyncTaskLoader.html) | Abstract loader that provides an [AsyncTask](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\AsyncTask.html) to do the work. |
| [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html) | A subclass of [AsyncTaskLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\AsyncTaskLoader.html) that queries the [ContentResolver](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ContentResolver.html) and returns a [Cursor](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\database\Cursor.html). This class implements the [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) protocol in a standard way for querying cursors, building on [AsyncTaskLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\AsyncTaskLoader.html) to perform the cursor query on a background thread so that it does not block the application's UI. Using this loader is the best way to asynchronously load data from a [ContentProvider](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ContentProvider.html), instead of performing a managed query through the fragment or activity's APIs. |

The classes and interfaces in the above table are the essential components you'll use to implement a loader in your application. You won't need all of them for each loader you create, but you'll always need a reference to the [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) in order to initialize a loader and an implementation of a[Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) class such as [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html). The following sections show you how to use these classes and interfaces in an application.

## Using Loaders in an Application

This section describes how to use loaders in an Android application. An application that uses loaders typically includes the following:

* An [Activity](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html) or [Fragment](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Fragment.html).
* An instance of the [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html).
* A [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html) to load data backed by a [ContentProvider](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ContentProvider.html). Alternatively, you can implement your own subclass of [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) or [AsyncTaskLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\AsyncTaskLoader.html) to load data from some other source.
* An implementation for [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html). This is where you create new loaders and manage your references to existing loaders.
* A way of displaying the loader's data, such as a [SimpleCursorAdapter](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\SimpleCursorAdapter.html).
* A data source, such as a [ContentProvider](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ContentProvider.html), when using a [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html).

### Starting a Loader

The [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) manages one or more [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) instances within an [Activity](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html) or [Fragment](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Fragment.html). There is only one [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) per activity or fragment.

You typically initialize a [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) within the activity's [onCreate()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle)) method, or within the fragment's[onActivityCreated()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Fragment.html#onActivityCreated(android.os.Bundle)) method. You do this as follows:

**// Prepare the loader.  Either re-connect with an existing one,  
// or start a new one.  
getLoaderManager().initLoader(0, null, this);**

The [initLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>)) method takes the following parameters:

* A unique ID that identifies the loader. In this example, the ID is 0.
* Optional arguments to supply to the loader at construction (null in this example).
* A [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) implementation, which the [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) calls to report loader events. In this example, the local class implements the [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) interface, so it passes a reference to itself, this.

The [initLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>)) call ensures that a loader is initialized and active. It has two possible outcomes:

* If the loader specified by the ID already exists, the last created loader is reused.
* If the loader specified by the ID does not exist, [initLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>)) triggers the[LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) method [onCreateLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onCreateLoader(int, android.os.Bundle)). This is where you implement the code to instantiate and return a new loader. For more discussion, see the section [onCreateLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\loaders.html#onCreateLoader).

In either case, the given [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) implementation is associated with the loader, and will be called when the loader state changes. If at the point of this call the caller is in its started state, and the requested loader already exists and has generated its data, then the system calls [onLoadFinished()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onLoadFinished(android.content.Loader<D>, D)) immediately (during [initLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>))), so you must be prepared for this to happen. See [onLoadFinished](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\loaders.html#onLoadFinished) for more discussion of this callback

Note that the [initLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>)) method returns the [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) that is created, but you don't need to capture a reference（引用）to it. **The**[**LoaderManager**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html)**manages the life of the loader automatically**. The [LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html) starts and stops loading when necessary, and maintains the state of the loader and its associated content. As this implies, you rarely interact with loaders directly (though for an example of using loader methods to fine-tune a loader's behavior, see the [LoaderThrottle](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\LoaderThrottle.html) sample). You most commonly use the [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) methods to intervene in the loading process when particular events occur. For more discussion of this topic, see [Using the LoaderManager Callbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\loaders.html#callback).

### Restarting a Loader

When you use [initLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>)), as shown above, it uses an existing loader with the specified ID if there is one. If there isn't, it creates one. But sometimes you want to discard your old data and start over.

To discard your old data, you use [restartLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#restartLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>)). For example, this implementation of[SearchView.OnQueryTextListener](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\SearchView.OnQueryTextListener.html) restarts the loader when the user's query changes. The loader needs to be restarted so that it can use the revised search filter to do a new query:

public boolean onQueryTextChanged(String newText) {  
    // Called when the action bar search text has changed.  Update  
    // the search filter, and restart the loader to do a new query  
    // with this filter.  
    mCurFilter = **!TextUtils.isEmpty(newText)** ? newText : null;  
    getLoaderManager().restartLoader(0, null, this);  
    return true;  
}

### Using the LoaderManager Callbacks

[LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) is a callback interface that lets a client interact with the[LoaderManager](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html).

Loaders, in particular [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html), **are expected to retain their data after being stopped**. This allows applications to keep their data across the activity or fragment's [onStop()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStop()) and [onStart()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Activity.html#onStart())methods, so that when users return to an application, they don't have to wait for the data to reload. You use the [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) methods when to know when to create a new loader, and to tell the application when it is time to stop using a loader's data.

[LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html) includes these methods:

* [onCreateLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onCreateLoader(int, android.os.Bundle)) — Instantiate and return a new [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) for the given ID.
* [onLoadFinished()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\LoaderManager.LoaderCallbacks.html" \l "onLoadFinished(android.content.Loader<D>, D)) — Called when a previously created loader has finished its load.
* [onLoaderReset()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onLoaderReset(android.content.Loader<D>)) — Called when a previously created loader is being reset, thus making its data unavailable.

These methods are described in more detail in the following sections.

#### onCreateLoader

**When you attempt to access a loader (for example, through**[**initLoader()**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.html#initLoader(int, android.os.Bundle, android.app.LoaderManager.LoaderCallbacks<D>))**), it checks to see whether the loader specified by the ID exists**. If it doesn't, it triggers the [LoaderManager.LoaderCallbacks](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html)method [onCreateLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onCreateLoader(int, android.os.Bundle)). This is where you create a new loader. Typically this will be a [**CursorLoader**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html)**,** but you can implement your own [Loader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Loader.html) subclass.

In this example, the [onCreateLoader()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\LoaderManager.LoaderCallbacks.html#onCreateLoader(int, android.os.Bundle)) callback method creates a[**CursorLoader**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html)**. You must build the**[**CursorLoader**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html)**using its constructor method, which requires the complete set of information needed to perform a query to the**[**ContentProvider**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ContentProvider.html). Specifically, it needs:

* uri — The URI for the content to retrieve.
* projection — A list of which columns to return. Passing null will return all columns, which is inefficient.
* selection — A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given URI.
* selectionArgs — You may include ?s in the selection, which will be replaced by the values fromselectionArgs, in the order that they appear in the selection. The values will be bound as Strings.
* sortOrder — How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.

For example:

 // If non-null, this is the current filter the user has provided.  
String mCurFilter;  
...  
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
    // This is called when a new Loader needs to be created.  This  
    // sample only has one Loader, so we don't care about the ID.  
    // First, pick the base URI to use depending on whether we are  
    // currently filtering.  
    Uri baseUri;  
    if (mCurFilter != null) {  
        baseUri = Uri.withAppendedPath(Contacts.CONTENT\_FILTER\_URI,  
                  Uri.encode(mCurFilter));  
    } else {  
        baseUri = Contacts.CONTENT\_URI;  
    }  
  
    // Now create and return a CursorLoader that will take care of  
    // creating a Cursor for the data being displayed.  
    String select = "((" + Contacts.DISPLAY\_NAME + " NOTNULL) AND ("  
            + Contacts.HAS\_PHONE\_NUMBER + "=1) AND ("  
            + Contacts.DISPLAY\_NAME + " != '' ))";  
    return new CursorLoader(getActivity(), baseUri,  
            CONTACTS\_SUMMARY\_PROJECTION, select, null,  
            Contacts.DISPLAY\_NAME + " COLLATE LOCALIZED ASC");  
}

#### onLoadFinished

This method is called **when a previously created loader has finished its load**. This method is guaranteed to be called prior to the release of the last data that was supplied for this loader. At this point you should remove all use of the old data (since it will be released soon), **but should not do your own release of the data since its loader owns it and will take care of that.**

The loader will release the data once it knows the application is no longer using it. For example, if the data is a cursor from a [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html), you should not call [close()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\database\Cursor.html#close()) on it yourself. **If the cursor is being placed in a**[**CursorAdapter**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\CursorAdapter.html)**, you should use the**[**swapCursor()**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\SimpleCursorAdapter.html#swapCursor(android.database.Cursor))**method so that the old** [**Cursor**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\database\Cursor.html)**is not closed.** For example:

// This is the Adapter being used to display the list's data.  
SimpleCursorAdapter mAdapter;  
...  
  
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {  
    // Swap the new cursor in.  (The framework will take care of closing the  
    // old cursor once we return.)  
    mAdapter.swapCursor(data);  
}

#### onLoaderReset

This method is called when a previously created loader is being reset, thus making its data unavailable. This callback lets you find out when the data is about to be released so you can remove your reference to it.

This implementation calls [swapCursor()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\SimpleCursorAdapter.html#swapCursor(android.database.Cursor)) with a value of null:

// This is the Adapter being used to display the list's data.  
SimpleCursorAdapter mAdapter;  
...  
  
public void onLoaderReset(Loader<Cursor> loader) {  
    // This is called when the last Cursor provided to onLoadFinished()  
    // above is about to be closed.  We need to make sure we are no  
    // longer using it.  
    **mAdapter.swapCursor(null);**  
}

## Example

As an example, here is the full implementation of a [Fragment](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Fragment.html) that displays a [ListView](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\widget\ListView.html) containing the results of a query against the contacts content provider. It uses a [CursorLoader](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\CursorLoader.html) to manage the query on the provider.

For an application to access a user's contacts, as shown in this example, its manifest must include the permission [READ\_CONTACTS](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\Manifest.permission.html#READ_CONTACTS).

public static class CursorLoaderListFragment extends ListFragment  
        implements OnQueryTextListener, LoaderManager.LoaderCallbacks<Cursor> {  
  
    // 用来展示list数据  
    SimpleCursorAdapter mAdapter;  
  
    // If non-null, this is the current filter the user has provided.  
    String mCurFilter;  
  
    @Override public void onActivityCreated(Bundle savedInstanceState) {  
        super.onActivityCreated(savedInstanceState);  
  
        // Give some text to display if there is no data.  In a real  
        // application this would come from a resource.  
        setEmptyText("No phone numbers");  
  
        // We have a menu item to show in action bar.  
        setHasOptionsMenu(true);  
  
        // Create an empty adapter we will use to display the loaded data.  
        mAdapter = new SimpleCursorAdapter(getActivity(),  
                android.R.layout.simple\_list\_item\_2, null,  
                new String[] { Contacts.DISPLAY\_NAME, Contacts.CONTACT\_STATUS },  
                new int[] { android.R.id.text1, android.R.id.text2 }, 0);  
        setListAdapter(mAdapter);  
  
        // Prepare the loader.  Either re-connect with an existing one,  
        // or start a new one.  
        getLoaderManager().initLoader(0, null, this);  
    }  
  
    @Override public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {  
        // Place an action bar item for searching.  
        MenuItem item = menu.add("Search");  
        item.setIcon(android.R.drawable.ic\_menu\_search);  
        item.setShowAsAction(MenuItem.SHOW\_AS\_ACTION\_IF\_ROOM);  
        SearchView sv = new SearchView(getActivity());  
        sv.setOnQueryTextListener(this);  
        item.setActionView(sv);  
    }  
  
    public boolean onQueryTextChange(String newText) {  
        // Called when the action bar search text has changed.  Update  
        // the search filter, and restart the loader to do a new query  
        // with this filter.  
        mCurFilter = !TextUtils.isEmpty(newText) ? newText : null;  
        getLoaderManager().restartLoader(0, null, this);  
        return true;  
    }  
  
    @Override public boolean onQueryTextSubmit(String query) {  
        // Don't care about this.  
        return true;  
    }  
  
    @Override public void onListItemClick(ListView l, View v, int position, long id) {  
        // Insert desired behavior here.  
        Log.i("FragmentComplexList", "Item clicked: " + id);  
    }  
  
    // These are the Contacts rows that we will retrieve.  
    static final String[] CONTACTS\_SUMMARY\_PROJECTION = new String[] {  
        Contacts.\_ID,  
        Contacts.DISPLAY\_NAME,  
        Contacts.CONTACT\_STATUS,  
        Contacts.CONTACT\_PRESENCE,  
        Contacts.PHOTO\_ID,  
        Contacts.LOOKUP\_KEY,  
    };  
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
        // This is called when a new Loader needs to be created.  This  
        // sample only has one Loader, so we don't care about the ID.  
        // First, pick the base URI to use depending on whether we are  
        // currently filtering.  
        Uri baseUri;  
        if (mCurFilter != null) {  
            baseUri = Uri.withAppendedPath(Contacts.CONTENT\_FILTER\_URI,  
                    Uri.encode(mCurFilter));  
        } else {  
            baseUri = Contacts.CONTENT\_URI;  
        }  
  
        // Now create and return a CursorLoader that will take care of  
        // creating a Cursor for the data being displayed.  
     String select = "((" + Contacts.DISPLAY\_NAME + " NOTNULL) AND ("  
                + Contacts.HAS\_PHONE\_NUMBER + "=1) AND ("  
                + Contacts.DISPLAY\_NAME + " != '' ))";  
        return new CursorLoader(getActivity(), baseUri,  
                CONTACTS\_SUMMARY\_PROJECTION, select, null,  
                Contacts.DISPLAY\_NAME + " COLLATE LOCALIZED ASC");  
    }  
  
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {  
        // Swap the new cursor in.  (The framework will take care of closing the  
        // old cursor once we return.)  
        mAdapter.swapCursor(data);  
    }  
  
    public void onLoaderReset(Loader<Cursor> loader) {  
        // This is called when the last Cursor provided to onLoadFinished()  
        // above is about to be closed.  We need to make sure we are no  
        // longer using it.  
        mAdapter.swapCursor(null);  
    }  
}

### More Examples

There are a few different samples in **ApiDemos** that illustrate how to use loaders:

* [LoaderCursor](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\LoaderCursor.html) — A complete version of the snippet shown above.
* [LoaderThrottle](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\LoaderThrottle.html) — An example of how to use throttling to reduce the number of queries a content provider does when its data changes.