The ContentProvider Class

Today’s Topics

ContentProvider

ContentResolver

CursorLoader

Implementing ContentProviders

ContentProvider

Represents a repository of structured data

Encapsulates data sets

Enforces data access permissions

ContentProvider

Intended for inter-application data sharing

Clients access ContentProviders through a ContentResolver

ContentResolver

Presents a database-style interface for reading & writing data

- query, insert, update, delete, etc.

Provides additional services such as change notification

ContentResolver

Get reference to ContentResolver by calling Context.getContentResolver()

ContentProvider & ContentResolver

Together these classes let code running in one process access data managed by another process

Android ContentProviders

Browser – bookmarks, history

Call log- telephone usage

Contacts – contact data

Media – media database

UserDictionary – database for predictive spelling

Many more

ContentProvider Data Model

Data represented logically as database tables

_ID	artist
13	Lady Gaga
44	Frank Sinatra
45	Elvis Presley
53	Barbara Streisand

URI

ContentProviders referenced by URIs

The format of the URI identifies specific data sets managed by specific ContentProviders

Format

content://authority/path/id

content – scheme indicating data that is managed by a content provider

authority – id for the content provider

path – 0 or more segments indicating the type of data to be accessed

id – a specific record being requested

Example: Contacts URI

ContactsContract.Contacts.CONTENT_URI =
“content://com.android.contacts/contacts/”

ContentResolver.query()

Returns a Cursor for iterating over a results set

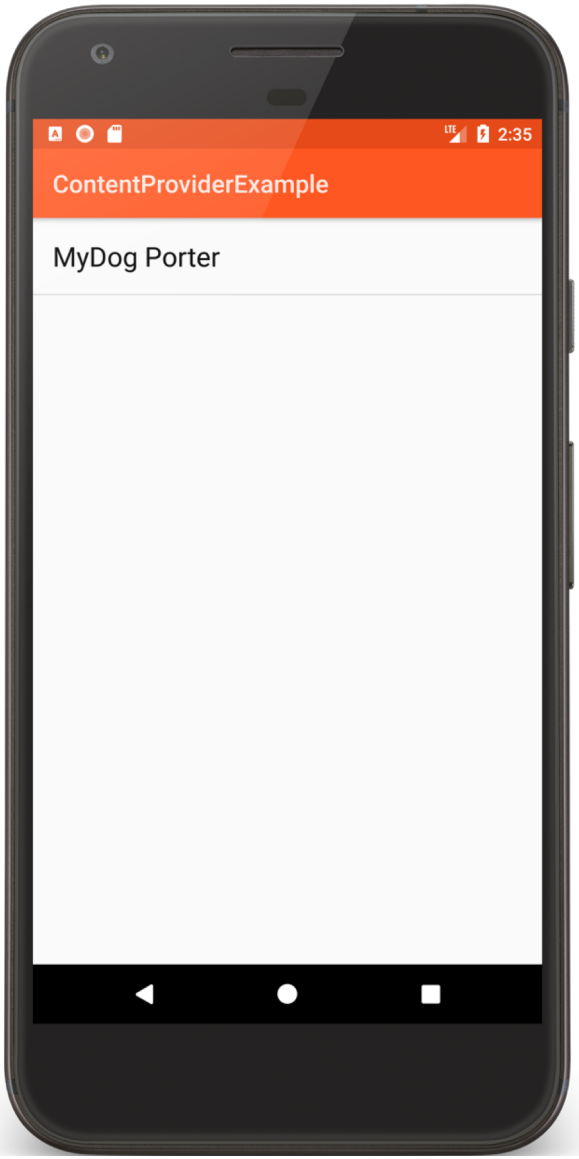
```
Cursor query (
    Uri uri,                    // ContentProvider Uri
    String[] projection         // Columns to retrieve
    String selection            // SQL selection pattern
    String[] selectionArgs      // SQL pattern args
    String sortOrder            // Sort order
    )
```

ContentProviderExample

Extracts Contact information from the Android Contacts ContentProvider

- Displays each contact’s name

ContentProvider Example



```
...
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      if (checkSelfPermission(Manifest.permission.READ_CONTACTS) !=
                                                        PackageManager.PERMISSION_GRANTED) {
         requestPermissions(permissions, mRequestCode);
      } else {
         displayContact();
      }
   }
```

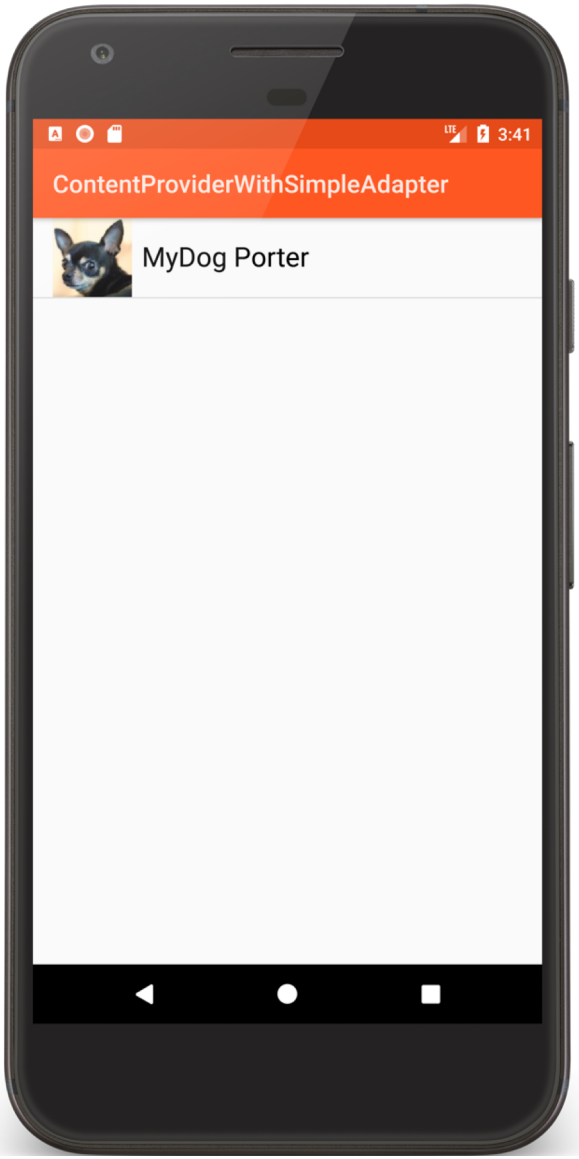
```
private void displayContact() {
   ContentResolver contentResolver = getContentResolver();
   Cursor cursor = contentResolver.query(ContactsContract.Contacts.CONTENT_URI,
                                 new String[]{ContactsContract.Contacts.DISPLAY_NAME},
                                  null, null, null);
   List<String> contacts = new ArrayList<>();
   if (null != cursor && cursor.moveToFirst()) {
      do {
           contacts.add(cursor.getString(cursor.getColumnIndex(
                                          ContactsContract.Contacts.DISPLAY_NAME)));
      } while (cursor.moveToNext());
      cursor.close();
   }
   ArrayAdapter<String> adapter =
                                 new ArrayAdapter<>(this, R.layout.list_item, contacts);
   setListAdapter(adapter);
}
```

ContentProviderWithSimpleAdapter

Extracts Contact information from the android Contacts ContentProvider

- Displays each contact’s name and photo, if available

ContentProvider
WithSimpleAdapter



```
private void loadContacts() {
  …
    // query contacts ContentProvider
    mCursor = contentResolver.query(Contacts.CONTENT_URI,
                                    columnsToExtract, whereClause, null, sortOrder);

    // pass cursor to custom list adapter
    setListAdapter(new ContactInfoListAdapter(this, R.layout.list_item, mCursor, 0));
```

```
public class ContactInfoListAdapter extends ResourceCursorAdapter {
  …
  public ContactInfoListAdapter(Context context, int layout, Cursor c, int flags) {
    ….
    // default thumbnail photo
    mNoPictureBitmap = (BitmapDrawable) context.getResources().getDrawable(
        R.drawable.ic_contact_picture, context.getTheme());
    mBitmapSize = (int) context.getResources().getDimension(
        R.dimen.textview_height);
    mNoPictureBitmap.setBounds(0, 0, mBitmapSize, mBitmapSize);
  }
```

```
// Create and return a new contact data view
public View newView(Context context, Cursor cursor, ViewGroup parent) {
  LayoutInflater inflater = (LayoutInflater) context
                                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
  return inflater.inflate(R.layout.list_item, parent, false);
}
```

```
// Update and return a contact data view
public void bindView(View view, Context context, Cursor cursor) {
  TextView textView = (TextView) view.findViewById(R.id.name);
  textView.setText(cursor.getString(cursor
                                    .getColumnIndex(Contacts.DISPLAY_NAME)));
  // Default photo
  BitmapDrawable photoBitmap = mNoPictureBitmap;
  // Get actual thumbnail photo if it exists
  String photoContentUri = cursor.getString(cursor .getColumnIndex(
                                      Contacts.PHOTO_THUMBNAIL_URI));
  if (null != photoContentUri) {
    InputStream input = null;
    try {
      // Read thumbnail data from input stream
      input = context.getContentResolver().openInputStream(
                                          Uri.parse(photoContentUri));
```

```
        if (input != null) {
          photoBitmap = new BitmapDrawable(
                                  mApplicationContext.getResources(), input);
          photoBitmap.setBounds(0, 0, mBitmapSize, mBitmapSize);
        }
      } …
    }

    // Set thumbnail image
    textView.setCompoundDrawables(photoBitmap, null, null, null);

  }
}
```

CursorLoader

Conducting intensive operations on the main thread can affect application responsiveness

CursorLoader uses an AsyncTask to perform queries on a background thread

Using a CursorLoader

Implement LoaderManager’s LoaderCallbacks interface

Create and initialize a cursor loader

initLoader()

Initialize and activate a Loader

```
Loader<D> initLoader(
                        int id,
                        Bundle args,
                        LoaderCallbacks<D> callback)
```

LoaderCallbacks

Called to instantiate and return a new Loader for the specified ID

```
Loader<D> onCreateLoader (
                                    int id,
                                    Bundle args)
```

LoaderCallbacks

Called when a previously created Loader has finished loading

```
void onLoadFinished(
                        Loader<D> loader,
                        D data)
```

LoaderCallbacks

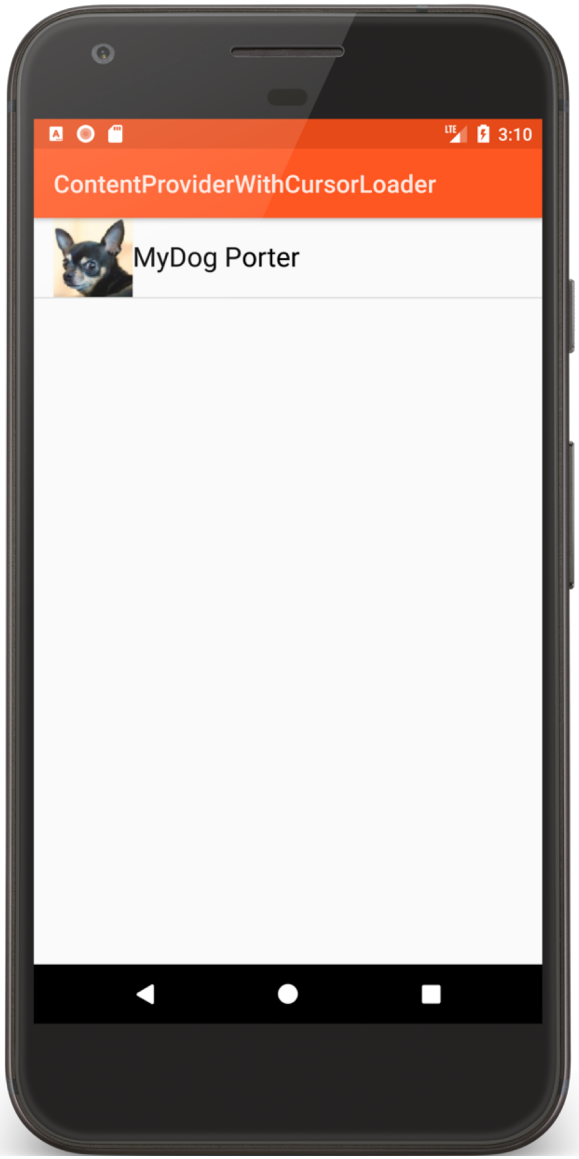
Called when a previously created Loader is reset

void onLoaderReset (Loader<D> loader)

ContentProviderWithCursorLoader

- Extracts Contact information from the Android Contacts ContentProvider
  - Displays each contact’s name and photo, if available
- Uses a CursorLoader when querying the ContentProvider

ContentProvider
WithCursorLoader



```
public class ContactsListActivity extends ListActivity implements
      LoaderManager.LoaderCallbacks<Cursor> {

   …
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      if (checkSelfPermission(Manifest.permission.READ_CONTACTS) !=
                                                      PackageManager.PERMISSION_GRANTED) {
         requestPermissions(permissions, mRequestCode);
      } else {
         loadContacts();
      }
   }
```

```
private void loadContacts() {
   // Create and set empty adapter
   mAdapter = new ContactInfoListAdapter(this, R.layout.list_item, null, 0);
   setListAdapter(mAdapter);
   // Initialize the loader
   getLoaderManager().initLoader(0, null, this);
}

// Contacts data items to extract
private static final String[] CONTACTS_ROWS = new String[]{Contacts._ID,
      Contacts.DISPLAY_NAME, Contacts.PHOTO_THUMBNAIL_URI};
```

```
// Called when a new Loader should be created. Returns a new CursorLoader
public Loader<Cursor> onCreateLoader(int id, Bundle args) {

   // String used to filter contacts with empty or missing names or are unstarred
   String select = "((" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
         + Contacts.DISPLAY_NAME + " != " ) AND (" + Contacts.STARRED
         + "== 1))";

   // String used for defining the sort order
   String sortOrder = Contacts._ID + " ASC";

   return new CursorLoader(this, Contacts.CONTENT_URI, CONTACTS_ROWS,
         select, null, sortOrder);
}
```

```
// Called when the Loader has finished loading its data
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
   // Swap the new cursor into the List adapter
   mAdapter.swapCursor(data);
}

// Called when the last Cursor provided to onLoadFinished()
// is about to be closed
public void onLoaderReset(Loader<Cursor> loader) {
   // set List adapter's cursor to null
   mAdapter.swapCursor(null);
}
```

ContentResolver.delete()

Returns the number of rows deleted

```
int delete (
 Uri url,                    // content Uri
 String where,               // SQL sel. pattern
 String[] selectArgs         // SQL pattern args
 )
```

ContentResolver.insert()

Returns the Uri of the inserted row

```
Uri insert (
    Uri url,                    // content Uri
    ContentValues values        // values
)
```

ContentResolver.update()

Returns the number of rows updated

```
int update(
    Uri url,                    // content Uri
    ContentValues values        // new field values
    String where,               // SQL sel. pattern
    String[] selectionArgs      // SQL pattern args
)
```

ContentProviderInsertContacts

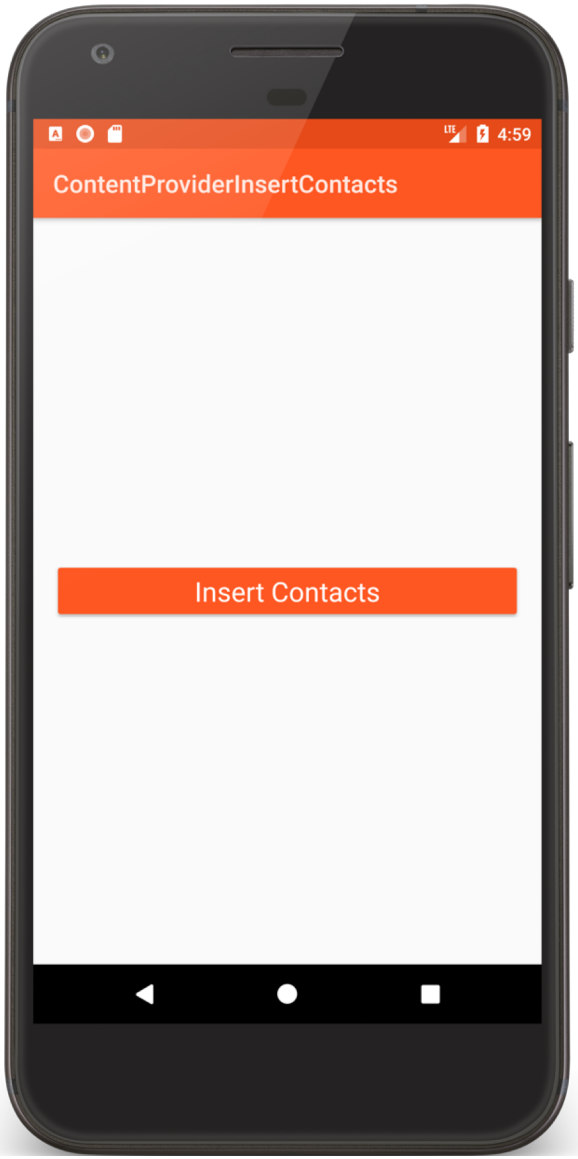
Application reads contact information from the Android Contacts ContentProvider

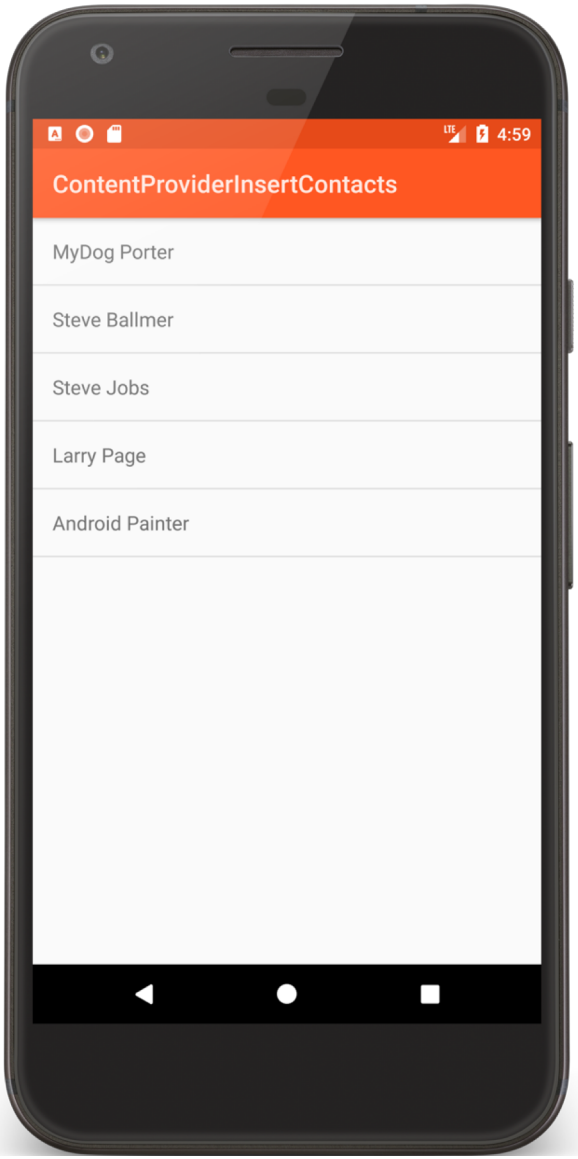
Inserts several new contacts into Contacts ContentProvider

Displays old and new contacts

Deletes these new contacts on exit

ContentProvider InsertContacts





```
public void onCreate(Bundle savedInstanceState) {
    ...
   // Get Account information
    mAccountList = AccountManager.get(this).getAccountsByType("com.google");
    // Must have a Google account set up on your device
    if (mAccountList.length == 0) finish();
    ...
    // Insert new contacts
    insertAllNewContacts();
    // Create and set empty list adapter
    mAdapter = new SimpleCursorAdapter(this, R.layout.list_layout, null,
          columnsToDisplay, resourceIds, 0);
    setListAdapter(mAdapter);

    // Initialize a CursorLoader
    getLoaderManager().initLoader(0, null, this);
  }
```

```
// Insert all new contacts into Contacts ContentProvider
private void insertAllNewContacts() {

    // Set up a batch operation on Contacts ContentProvider
    ArrayList<ContentProviderOperation> batchOperation = new
                                                ArrayList<ContentProviderOperation>();
    for (String name : mNames) {
        addRecordToBatchInsertOperation(name, batchOperation);
    }
    try {
        // Apply all batched operations
        getContentResolver().applyBatch(ContactsContract.AUTHORITY, batchOperation);
    }
  ...
```

```
// Insert named contact into Contacts ContentProvider
private void addRecordToBatchInsertOperation(String name,
                                             List<ContentProviderOperation> ops) {
   int position = ops.size();
   // First part of operation
   ops.add(ContentProviderOperation.newInsert(RawContacts.CONTENT_URI)
         .withValue(RawContacts.ACCOUNT_TYPE, mType)
         .withValue(RawContacts.ACCOUNT_NAME, mName)
         .withValue(Contacts.STARRED, 1).build());

   // Second part of operation
   ops.add(ContentProviderOperation.newInsert(Data.CONTENT_URI)
         .withValueBackReference(Data.RAW_CONTACT_ID, position)
         .withValue(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE)
         .withValue(StructuredName.DISPLAY_NAME, name).build());
}
```

```
private void deleteAllNewContacts() {
    for (String name : mNames) {
        deleteContact(name);
    }
}

private void deleteContact(String name) {
    getContentResolver().delete(ContactsContract.RawContacts.CONTENT_URI,
            ContactsContract.Contacts.DISPLAY_NAME + "=?", new String[]{name});
}
```

Creating a ContentProvider

- Implement a storage system for the data
- Define a Contract Class to support users of your ContentProvider
- Implement a ContentProvider subclass
- Declare and configure ContentProvider in AndroidManifest.xml

ContentProviderCustom

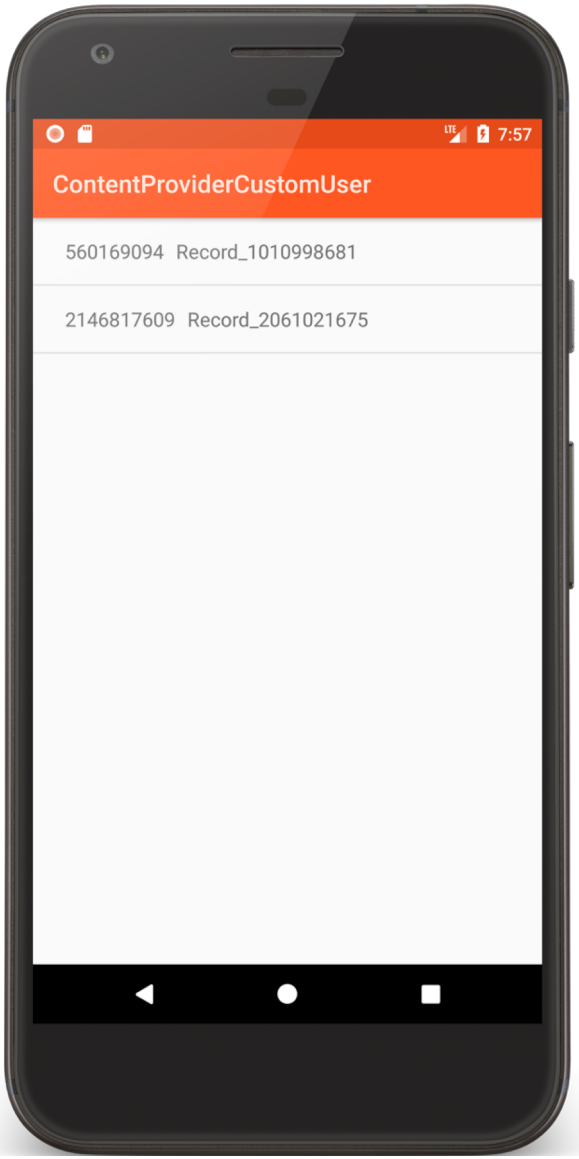
Application defines a ContentProvider for ID/string pairs

ContentProviderCustomUser

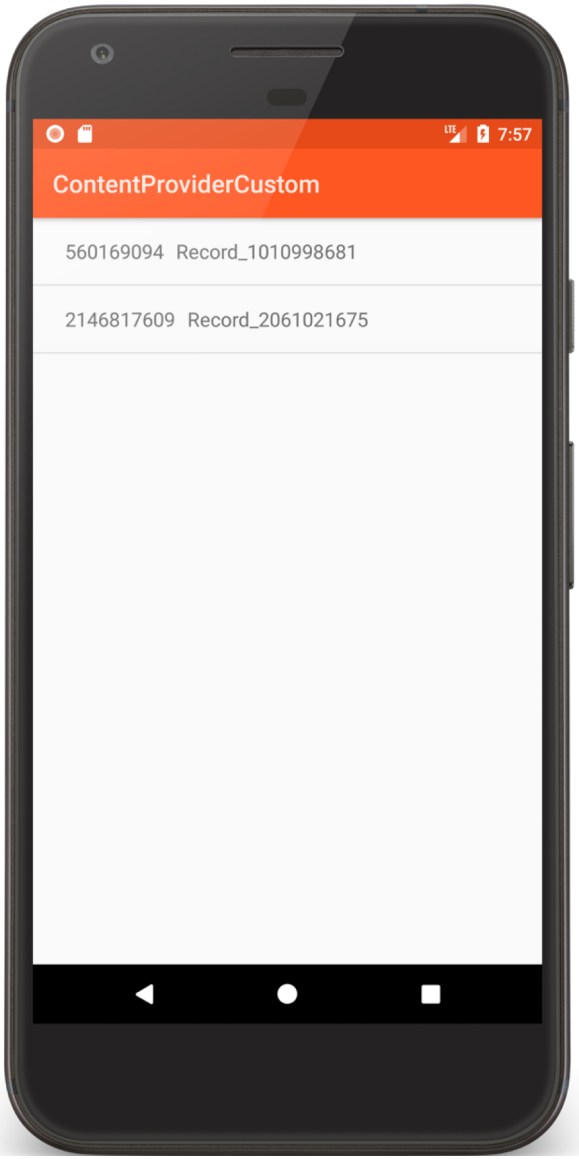
Reads ID/String pairs from the ContentProvider

Inserts and deletes ID/String pairs

ContentProvider
CustomUser



ContentProvider Custom



Next Time

The Service class

Example Applications

ContentProviderExample

ContentProviderWithSimpleAdapter

ContentProviderWithCursorLoader

ContentProviderInsertContacts

ContentProviderCustom

ContentProviderCustomUser