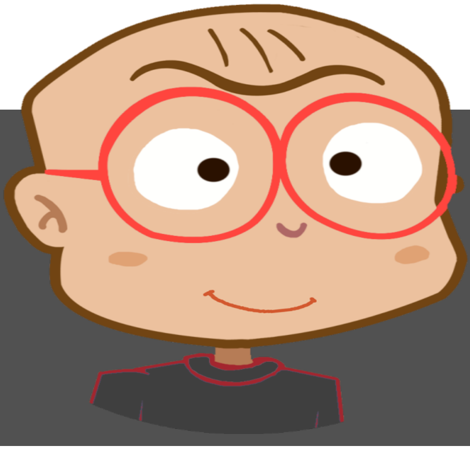


LESSON 10 – Content Provider

START »

Learning Goals

After the course, attendees will be able to:

- Have basic understanding about Android's Content Provider
- Have basic understanding about the ways to access data in a content provider
- Have basic ability to create a custom Content Provider

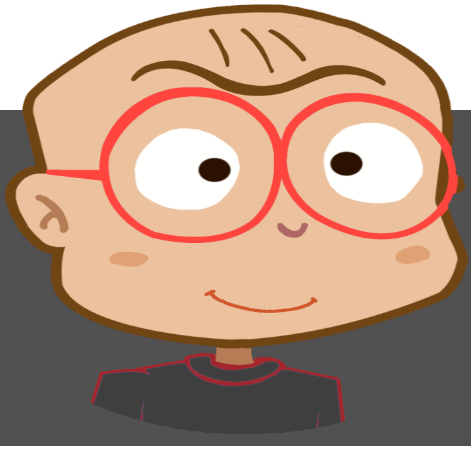
Table of contents

Overview

- Content providers manage access to a structured set of data.
- They encapsulate the data, and provide mechanisms for defining data security.
- Content providers are the standard interface that connects data in one process with code running in another process.

Overview

- When do we need to develop our own provider:
  - To share our data with other applications.
  - To provide custom search suggestions in our own application.
  - To copy & paste complex data or files from our application to other applications.
- Android itself includes content providers that manage data such as audio, video, images, and personal contact information. Refer to: android.provider package.



Content Provider Basics

START »

Content Provider Basics

This topic describes the basics of the following:

- How content providers work.
- The API you use retrieve data from a content provider.
- The API you use to insert, update, or delete data in a content provider.
- Other API features that facilitate working with providers.

Overview

- A content provider presents data to external applications as one or more tables are similar to the tables found in a relational database:
  - A row represents an instance of some type of data the provider collects.
  - Each column in the row represents an individual piece of data collected for an instance.

stu_name	stu_acc	stu_class	stu_mark	_ID
Maria Ozawa	MariO	JAV	9	1

Overview/Accessing a provider

- An application accesses the data from a content provider with a ContentResolver client object:
  - This object has methods that call identically-named methods in the provider object, an instance of one of the concrete subclasses of ContentProvider.
  - The ContentResolver methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.
  - To access a provider, your application usually has to request specific permissions in its manifest file.

Overview/Accessing a provider

- For example, to get a list of the words and their locales from the User Dictionary Provider:
  - We call **ContentResolver.query()**.
  - The query() method calls the **ContentProvider.query()** method defined by the User Dictionary Provider.

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```

Overview/Accessing a provider

- Query() compared to SQL query:

query() argument	SELECT keyword/parameter	Notes
Uri	FROM *table_name*	Uri maps to the table in the provider named table_name.
projection	*col,col,col,...*	projection is an array of columns that should be included for each row retrieved.
selection	WHERE *col* = *value*	selection specifies the criteria for selecting rows.
selectionArgs	No exact equivalent.	Selection arguments replace ? placeholders in the selection clause.
sortOrder	ORDER BY *col,col,...*	sortOrder specifies the order in which rows appear in the returned Cursor.

Overview/Content URIs

- A content URI is a URI that identifies data in a provider. Content URIs include the symbolic name of the entire provider (its authority) and a name that points to a table (a path):
  - The ContentResolver object parses out the URI's authority, and uses it to "resolve" the provider by comparing the authority to a system table of known providers.
  - The ContentResolver can then dispatch the query arguments to the correct provider.
  - The ContentProvider uses the path part of the content URI to choose the table to access. A provider usually has a path for each table it exposes.

Overview/Content URIs

- For example: **content://**user_dictionary/words. In which:
  - user_dictionary string is the provider's authority.
  - words string is the table's path.
  - The string **content://** (the scheme) is always present, and identifies this as a content URI.
- Many providers allow you to access a single row in a table by appending an ID value to the end of the URI.
  - Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);

Retrieving Data from the Provider

- To retrieve data from a provider, follow these basic steps:
    - Request the read access permission for the provider.
        - You can't request this permission at run-time.
        - Instead, you have to specify that you need this permission in your manifest, using the **<uses-permission>** element and the exact permission name defined by the provider.
    - Define the code that sends a query to the provider.

Retrieving Data from the Provider

- Constructing the query:
  - SELECT _ID, word, locale FROM words WHERE word = <userinput> ORDER BY word ASC;
- Protecting against malicious input
  - String **mSelectionClause** = "var = " + mUserInput;
  - What if mUserInput = "var = nothing; DROP TABLE *;”;
  - Instead of using concatenation to include the user input, use this:
    - String **mSelectionClause** = "var = ?";
    - String[] **selectionArgs** = {""};
    - selectionArgs[0] = mUserInput;

Content Provider Permissions

- A provider's application can specify permissions that other applications must have in order to access the provider's data.
  - End users see the requested permissions when they install the application.
- If a provider's application doesn't specify any permissions, then other applications have no access to the provider's data.
  - However, components in the provider's application always have full read and write access.
- To get the permissions, an application requests them with a <uses-permission> element in its manifest file.
  - <uses-permission android:name="android.permission.READ_USER_DICTIONARY">

Inserting, Updating, and Deleting Data

- To insert data into a provider, call the **ContentResolver.insert()**:
  - Defines an object to contain the new values to insert:

    ContentValues mNewValues = new **ContentValues()**;
  - Sets the values of each column and inserts the word:

    mNewValues.**put**(UserDictionary.Words.APP_ID, "example.user");

    mNewValues.**put**(UserDictionary.Words.LOCALE, "en_US");

    mNewValues.**put**(UserDictionary.Words.WORD, "insert");

    mNewValues.**put**(UserDictionary.Words.FREQUENCY, "100");
  - Uri uri = getContentResolver().**insert**(UserDictionary.Word.CONTENT_URI, mNewValues);

    // returned uri = content://user_dictionary/words/<id_value>

Inserting, Updating, and Deleting Data

- To update a row, you use a ContentValues object with the updated values just as you do with an insertion, and selection criteria just as you do with a query.
- The client method you use is **ContentResolver.update()**:
  - Defines selection criteria for the rows you want to update

    String **mSelectionClause** = UserDictionary.Words.LOCALE + "LIKE ?”;

    String[] **mSelectionArgs** = {"en_%"};
  - Defines an object to contain the updated values

    ContentValues **mUpdateValues** = new ContentValues();

    mUpdateValues.**putNull**(UserDictionary.Words.LOCALE);
  - mRowsUpdated = getContentResolver().**update**(UserDictionary.Words.

    CONTENT_URI, mUpdateValues, mSelectionClause , mSelectionArgs);

Inserting, Updating, and Deleting Data

- Deleting rows is similar to retrieving row data: you specify selection criteria for the rows you want to delete and the client method returns the number of deleted rows:
  - Defines selection criteria for the rows you want to delete

    String **mSelectionClause** = UserDictionary.Words.APP_ID + " LIKE ?”;

    String[] **mSelectionArgs** = {"user"};
  - Defines a variable to contain the number of rows deleted

    int mRowsDeleted = 0;
  - Deletes the words that match the selection criteria

    mRowsDeleted = getContentResolver().**delete**(

    UserDictionary.Words.CONTENT_URI, mSelectionClause, mSelectionArgs);

Alternative Forms of Provider Access

- Three alternative forms of provider access are important in application development:
    - **Batch access**: You can create a batch of access calls with methods in the **ContentProviderOperation** class, and then apply them with **ContentResolver.applyBatch()**.
    - **Asynchronous queries**: You should do queries in a separate thread. One way to do this is to use a **CursorLoader** object.
    - **Data access via intents**: Although you can't send an intent directly to a provider, you can send an intent to the provider's application, which is usually the best-equipped to modify the provider's data.

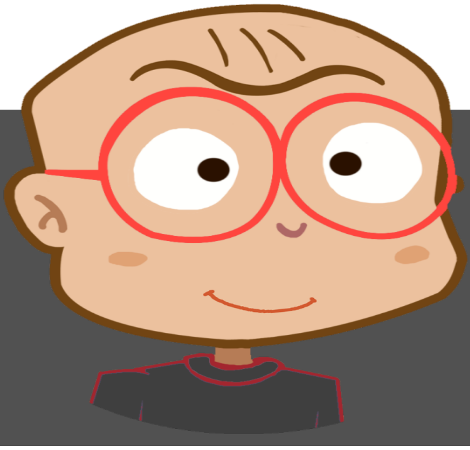
Contract Classes

- A contract class defines constants that help applications work with the content URIs, column names, intent actions, and other features of a content provider:

```
public final class MyContract {
   public static final String AUTHORITY = "authority";
   public static final String AUTHORITY_URI = "content://authority";

   public MyContract() {}

   public static abstract class MyTable implements BaseColumns {
      public static final String TABLE_NAME = "table_name";
       ...
   }
}
```



Creating a Content Provider

START »

Before You Start Building

- Decide if you need a content provider:
  - You need to build a content provider if you want to provide one or more of the following features:
    - You want to offer complex data or files to other applications.
    - You want to allow users to copy complex data from your app into other apps.
    - You want to provide custom search suggestions using the search framework.

Designing Data Storage

- Before you create the interface, you must decide how to store the data.
- A content provider is the interface to data saved in a structured format.
- These are some of the data storage technologies that are available in Android:
    - SQLite database API: is used to store table-oriented data.
    - File-oriented APIs: is used to store file data.
    - java.net and android.net: is used to store network-based data.

Designing Data Storage

- Some tips for designing your provider's data structure:
  - Table data should always have a "primary key" column that the provider maintains as a unique numeric value for each row.
  - With file-oriented data: Store the data in a file and then provide it indirectly rather than storing it directly in a table.
  - With the Binary Large OBject (BLOB) data type, you define a primary key column, a MIME type column, and one or more generic columns as BLOB.

Designing Content URIs

- Designing an authority:
  - com.example.<appname>.provider
- Designing a path structure:
  - com.example.<appname>.provider/table1
- Content URI patterns:
  - *: Matches a string of any valid characters of any length.
    - content://com.example.app.provider/*
  - #: Matches a string of numeric characters of any length.
    - content://com.example.app.provider/table3/6
    - Matches a content URI for the row identified by 6

Implementing the ContentProvider Class

- The abstract class ContentProvider defines six abstract methods:
  - **query()** *// returns the data as a Cursor object*
    - Retrieve data from your provider. Use the arguments to select the table to query, the rows and columns to return, and the sort order of the result.
  - **insert()** *// returns a content URI for the newly-inserted row.*
    - Insert a new row into your provider. Use the arguments to select the destination table and to get the column values to use.
  - **update()** *// returns the number of rows updated.*
    - Update existing rows in your provider. Use the arguments to select the table and rows to update and to get the updated column values.

Implementing the ContentProvider Class

- The abstract class ContentProvider defines six abstract methods:
  - **delete()** *// returns the number of rows deleted.*
    - Delete rows from your provider. Use the arguments to select the table and the rows to delete.
  - **getType()** *// returns the MIME type corresponding to a content URI.*
    - This method is described in more detail in the section Implementing Content Provider MIME Types.
  - **onCreate()**
    - Initialize your provider. The Android system calls this method immediately after it creates your provider. Notice that your provider is not created until a ContentResolver object tries to access it.

Implementing Content Provider MIME Types

- The ContentProvider class has two methods for returning MIME types:
  - **getType()**
    - One of the required methods that you must implement for any provider.
  - **getStreamTypes()**
    - A method that you're expected to implement if your provider offers files.

Implementing a Contract Class

- A contract class is a public final class that contains constant definitions for the URIs, column names, MIME types, and other meta-data that pertain to the provider:

```
public final class MyContract {
   public static final String AUTHORITY = "authority";
   public static final String AUTHORITY_URI = "content://authority";

   public MyContract() {}

   public static abstract class MyTable implements BaseColumns {
      public static final String TABLE_NAME = "table_name";
       ...
   }
}
```

Implementing Content Provider Permissions

- More fine-grained permissions take precedence over ones with larger scope:
    - Single read-write provider-level permission:
        - One permission that controls both read and write access to the entire provider, specified with the android:permission attribute of the <provider> element.
    - Separate read and write provider-level permission.
        - A read permission and a write permission for the entire provider.

The <provider> Element

- Like Activity and Service components, a subclass of ContentProvider must be defined in the manifest file for its application, using the <provider> element:
    - Authority (android:authorities)
    - Provider class name (android:name)
    - Permissions
    - Startup and control attributes
    - Informational attributes

Intents and Data Access

- Applications can access a content provider indirectly with an Intent. The application does not call any of the methods of ContentResolver or ContentProvider.
- Instead, it sends an intent that starts an activity, which is often part of the provider's own application.
- The destination activity is in charge of retrieving and displaying the data in its UI.

Summary

Content Provider Basics	Creating a Content Provider
Learned: How to access data in a content provider when the data is organized in tables.	How to create your own content provider.

Exit Course

THANK YOU

EXIT