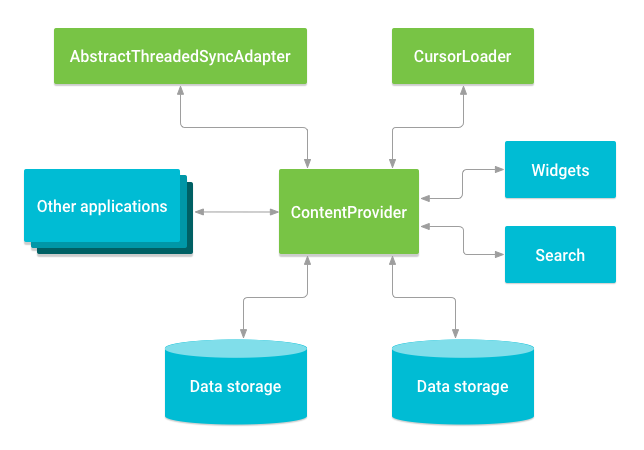
**UNIT 3 - MOBILE DATA MANAGEMENT**

**CONTENT PROVIDER**

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

A content provider coordinates access to the data storage layer in your application for a number of different APIs and components as illustrated in figure 1, these include:

* Sharing access to your application data with other applications
* Sending data to a widget
* Returning custom search suggestions for your application through the search framework using [SearchRecentSuggestionsProvider](https://developer.android.com/reference/android/content/SearchRecentSuggestionsProvider.html)
* Synchronizing application data with your server using an implementation of [AbstractThreadedSyncAdapter](https://developer.android.com/reference/android/content/AbstractThreadedSyncAdapter.html)
* Loading data in your UI using a [CursorLoader](https://developer.android.com/reference/android/content/CursorLoader.html)

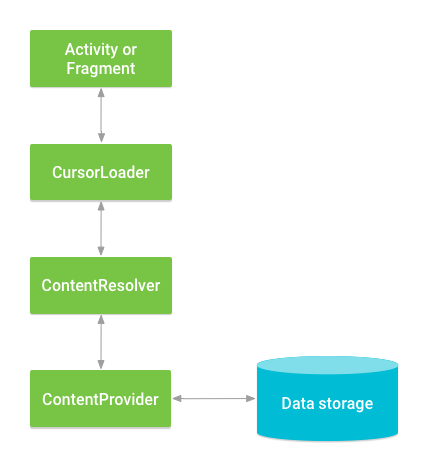


**Figure 1.** Relationship between content provider and other components.

**Accessing a provider**

When you want to access data in a content provider, you use the [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) object in your application's [Context](https://developer.android.com/reference/android/content/Context.html) to communicate with the provider as a client. The [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) object communicates with the provider object, an instance of a class that implements [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html). The provider object receives data requests from clients, performs the requested action, and returns the results. This object has methods that call identically-named methods in the provider object, an instance of one of the concrete subclasses of [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html). The [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

A common pattern for accessing a [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) from your UI uses a [CursorLoader](https://developer.android.com/reference/android/content/CursorLoader.html) to run an asynchronous query in the background. The [Activity](https://developer.android.com/reference/android/app/Activity.html) or [Fragment](https://developer.android.com/reference/android/app/Fragment.html) in your UI call a [CursorLoader](https://developer.android.com/reference/android/content/CursorLoader.html) to the query, which in turn gets the [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) using the [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html). This allows the UI to continue to be available to the user while the query is running. This pattern involves the interaction of a number of different objects, as well as the underlying storage mechanism, as illustrated in figure 2.



**Figure 2.** Interaction between ContentProvider, other classes, and storage.

One of the built-in providers in the Android platform is the user dictionary, which stores the spellings of non-standard words that the user wants to keep. Table 1 illustrates what the data might look like in this provider's table:

**Table 1:** Sample user dictionary table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| word | app id | frequency | locale | \_ID |
| mapreduce | user1 | 100 | en\_US | 1 |
| precompiler | user14 | 200 | fr\_FR | 2 |
| applet | user2 | 225 | fr\_CA | 3 |
| const | user1 | 255 | pt\_BR | 4 |
| int | user5 | 100 | en\_UK | 5 |

In table 1, each row represents an instance of a word that might not be found in a standard dictionary.

Each column represents some data for that word, such as the locale in which it was first encountered. The column headers are column names that are stored in the provider. To refer to a row's locale, you refer to its locale column. For this provider, the \_ID column serves as a "primary key" column that the provider automatically maintains.

To get a list of the words and their locales from the User Dictionary Provider, you call [ContentResolver.query()](https://developer.android.com/reference/android/content/ContentResolver.html#query(android.net.Uri,%20java.lang.String[],%20android.os.Bundle,%20android.os.CancellationSignal)). The [query()](https://developer.android.com/reference/android/content/ContentResolver.html#query(android.net.Uri,%20java.lang.String[],%20android.os.Bundle,%20android.os.CancellationSignal)) method calls the [ContentProvider.query()](https://developer.android.com/reference/android/content/ContentProvider.html#query(android.net.Uri,%20java.lang.String[],%20android.os.Bundle,%20android.os.CancellationSignal)) method defined by the User Dictionary Provider. The following lines of code show a [ContentResolver.query()](https://developer.android.com/reference/android/content/ContentResolver.html#query(android.net.Uri,%20java.lang.String[],%20android.os.Bundle,%20android.os.CancellationSignal)) call:

cursor = getContentResolver().query(  
    UserDictionary.Words.CONTENT\_URI , projection ,  
    selectionClause,selectionArgs, sortOrder);

Table 2 shows how the arguments to [query(Uri,projection,selection,selectionArgs,sortOrder)](https://developer.android.com/reference/android/content/ContentResolver.html#query(android.net.Uri,%20java.lang.String[],%20android.os.Bundle,%20android.os.CancellationSignal)) match an SQL SELECT statement:

**Table 2:** 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](https://developer.android.com/reference/android/database/Cursor.html). |

**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**). When you call a client method to access a table in a provider, the content URI for the table is one of the arguments.

In the preceding lines of code, the constant [CONTENT\_URI](https://developer.android.com/reference/android/provider/UserDictionary.Words.html#CONTENT_URI) contains the content URI of the user dictionary's "words" table. The [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) 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](https://developer.android.com/reference/android/content/ContentResolver.html) can then dispatch the query arguments to the correct provider.

The [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) 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.

In the previous lines of code, the full URI for the "words" table is:

content://user\_dictionary/words

**CRUD – Create , Retrieve, Update, Delete**

### Inserting data

To insert data into a provider, you call the [ContentResolver.insert()](https://developer.android.com/reference/android/content/ContentResolver.html#insert(android.net.Uri,%20android.content.ContentValues)) method. This method inserts a new row into the provider and returns a content URI for that row. This snippet shows how to insert a new word into the User Dictionary Provider:

[JAVA](https://developer.android.com/guide/topics/providers/content-provider-basics#java)

Uri newUri;  
  
ContentValues newValues = new ContentValues();  
  
newValues.put(UserDictionary.Words.APP\_ID, "example.user");  
newValues.put(UserDictionary.Words.LOCALE, "en\_US");  
newValues.put(UserDictionary.Words.WORD, "insert");  
newValues.put(UserDictionary.Words.FREQUENCY, "100");  
  
newUri = getContentResolver().insert(  
    UserDictionary.Words.CONTENT\_URI,     
    newValues                             
);

The data for the new row goes into a single [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) object, which is similar in form to a one-row cursor. The columns in this object don't need to have the same data type, and if you don't want to specify a value at all, you can set a column to null using [ContentValues.putNull()](https://developer.android.com/reference/android/content/ContentValues.html#putNull(java.lang.String)).

The snippet doesn't add the \_ID column, because this column is maintained automatically. The provider assigns a unique value of \_ID to every row that is added. Providers usually use this value as the table's primary key.

### Updating data

To update a row, you use a [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) 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()](https://developer.android.com/reference/android/content/ContentResolver.html#update(android.net.Uri,%20android.content.ContentValues,%20java.lang.String,%20java.lang.String[])). You only need to add values to the [ContentValues](https://developer.android.com/reference/android/content/ContentValues.html) object for columns you're updating. If you want to clear the contents of a column, set the value to null.

The following snippet changes all the rows whose locale has the language "en" to a have a locale of null. The return value is the number of rows that were updated:

[JAVA](https://developer.android.com/guide/topics/providers/content-provider-basics#java)

ContentValues updateValues = new ContentValues();  
  
  
String selectionClause = UserDictionary.Words.LOCALE +  " LIKE ?";  
String[] selectionArgs = {"en\_%"};  
  
int rowsUpdated = 0;  
  
updateValues.putNull(UserDictionary.Words.LOCALE);  
  
rowsUpdated = getContentResolver().update(  
    UserDictionary.Words.CONTENT\_URI,     
    updateValues selectionClause,selectionArgs                         
);

### 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. The following snippet deletes rows whose appid matches "user". The method returns the number of deleted rows.

[JAVA](https://developer.android.com/guide/topics/providers/content-provider-basics#java)

String selectionClause = UserDictionary.Words.APP\_ID + " LIKE ?";  
String[] selectionArgs = {"user"};  
  
int rowsDeleted = 0;  
rowsDeleted = getContentResolver().delete(  
    UserDictionary.Words.CONTENT\_URI,      selectionClause,  selectionArgs                         
);

### STANDARD CONTENT PROVIDERS

A number of content providers are part of Android's API. All these standard providers are defined in the package android.provider. The following table lists the standard providers and what they are used for.

|  |  |  |
| --- | --- | --- |
| Provider | Since | Usage |
| Browser | SDK 1 | Manages your web-searches, bookmarks and browsing-history. |
| CalendarContract | SDK 14 | Manages the calendars on the user's device. |
| CallLog | SDK 1 | Keeps track of your call history. |
| Contacts | SDK 1 | The old and deprecated content provider for managing contacts. You should only use this provider if you need to support an SDK prior to SDK 5! |
| ContactsContract | SDK 5 | Deals with all aspects of contact management. Supersedes the Contacts-content provider. |
| MediaStore | SDK 1 | The content provider responsible for all your media files like music, video and pictures. |
| Settings | SDK 1 | Manages all global settings of your device. |
| UserDictionary | SDK 3 | Keeps track of words you add to the default dictionary. |

**Browser**

Browser content provider exposes the user’s browser site history and their bookmarked websites. You access this content provider via the android.provider.Browser class. As with the CallLog class, you can use the information provided by the Browser content provider to generate statistics and to provide cross-application functionality. You might use the Browser content provider to add a bookmark for your application support website.

**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 }  
 **private void** fetchbrowser()  
 {  
 Uri uri= Browser.BookmarkColumns.CONTENT\_URI;  
 String[] projection= { Browser.BookmarkColumns.TITLE, Browser.BookmarkColumns.VISITS };  
 String selction = **null**;  
 String[] selctionArgs=**null**;  
 String sortOrder=**null**;  
 ContentResolver resolver = getContentResolver();  
 Cursor cursor= resolver.query (uri ,projection, selction, selctionArgs, sortOrder);  
  
 }

**Contact**

Contacts database is one of the most commonly used applications on the mobile phone. People always want phone numbers handy for calling friends, family, coworkers, and clients. Additionally, most phones show the identity of the caller based on the contacts application, including nicknames, photos, or icons.

Android provides a built-in Contact application, and the contact data is exposed to other Android applications using the content provider interface. As an application developer, this means you can leverage the user’s contact data within your application for a more robust user experience.

**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 fetchContact();  
 }  
  
  
 **private void** fetchContact()  
 {  
  
 Uri uri= ContactsContract.CommonDataKinds.Phone.CONTENT\_URI;  
 ContentResolver resolver= getContentResolver();  
 String[] projection={ContactsContract.CommonDataKinds.Phone.DISPLAY\_NAME, ContactsContract.CommonDataKinds.Phone.NUMBER};  
 String selection = **null**;  
 String[] selectionArgs=**null**;  
 String sortOrder=**null**;  
 Cursor cursor= resolver.query(uri,projection,selection, selectionArgs,sortOrder);  
 **while** (cursor.moveToNext()){  
  
 String name= cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.DISPLAY\_NAME));  
 String number=cursor.getString(cursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));  
 Log.d(**"AOS"**,**"NAME"** +name+ **"NUMBER"**+number);  
  
 }

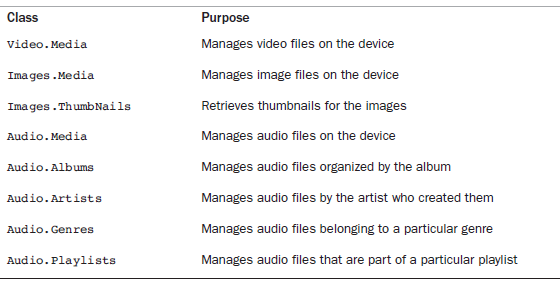
**Media store**

MediaStore content provider to access media on the phone and on external storage devices. The primary types of media that you can access are audio, images, and video. You can access these different types of media through their respective content provider classes under android.provider.MediaStore.

Most of the MediaStore classes allow full interaction with the data. You can retrieve, add, and delete media files from the device. There are also a handful of helper classes that define the most common data columns that can be requested.

Table lists some commonly used classes that you can find under android. provider. MediaStore.

**Common MediaStore Classes**



**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 }  
  
**private void** fetchMedia()  
{  
 Uri uri= MediaStore.Audio.Media.INTERNAL\_CONTENT\_URI;  
 String[] projection = {MediaStore.Audio.Media.TITLE, MediaStore.Audio.Media.DURATION};  
 String selection = **null**;  
 String selectionArgs[]= **null**;  
 String sortOrder= **null**;  
 ContentResolver resolver= getContentResolver();  
 Cursor cursor=resolver.query(uri,projection,selection,selectionArgs,sortOrder);  
 **while**(cursor.moveToNext()){  
 String name= cursor.getColumnName(cursor.getColumnIndex(MediaStore.Audio.Media.TITLE));  
 String duration= cursor.getColumnName(cursor.getColumnIndex(MediaStore.Audio.Media.DURATION));  
 Log.d(**"AOS"**,**"NAME"**+name+**"D"**+duration);  
  
  
 }

**Call log**

Android provides a content provider to access the call log on the handset via the class android.provider.CallLog. At first glance, the CallLog might not seem to be a useful provider for developers, but it has some nifty features. You can use the CallLog to filter recently dialed calls, received, and missed calls. The date and duration of each call is logged and tied back to the Contact application for caller identification purposes.

The CallLog is a useful content provider for customer relationship management (CRM) applications. The user can also tag specific phone numbers with custom labels within the Contact application.

**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 }  
 **private void** fetchcall()  
 {  
 Uri uri=CallLog.Calls.CONTENT\_URI;  
 String[] projection= {CallLog.Calls.CACHED\_NUMBER\_LABEL,CallLog.Calls.DURATION};  
 String selction = **null**;  
 String[] selctionArgs=**null**;  
 String sortOrder=**null**;  
 ContentResolver resolver = getContentResolver();  
 Cursor cursor= resolver.query (uri ,projection, selction, selctionArgs, sortOrder);  
 **while** (cursor.moveToNext()){  
  
 String label= cursor.getColumnName(cursor.getColumnIndex(CallLog.Calls.CACHED\_NUMBER\_LABEL);  
 String duration = cursor.getColumnName(cursor.getColumnIndex(CallLog.Calls.DURATION));  
  
  
 }

**Using the UserDictionary Content Provider**

Another useful content provider is the UserDictionary provider. You can use this content provider for predictive text input on text fields and other user input mechanisms. Individual words stored in the dictionary are weighted by frequency and organized by locale. You can use the addWord() method within the UserDictionary. Words class to add words to the custom user dictionary.

**Using the Settings Content Provider**

Another useful content provider is the Settings provider. You can use this content provider to access the device settings and user preferences. Settings are organized much as they are in the Settings application—by category. You can find information about the Settings content provider in the android.provider.Settings class.

**Data Access and storage**

* **Shared Preferences** — Store private primitive data in key-value pairs.
* **Internal Storage**— Store private data on the device memory.
* **External Storage** — Store public data on the shared external storage.
* **SQLite Databases** — Store structured data in a private database.
* **Network Connection** — Store data on the web with your own network server.

#### Internal Storage

When you install some application on your phone, then the Android system will provide you with some kind of private internal storage where the application can store its private data. This data can't be accessed by any other application. When you uninstall the application, then all the data related to that application will be deleted.

**Use Internal Storage**

When you want to have some private data for your application, then you can use internal storage. Another thing that is to be noted here is that if your application is storing some data that can be used by other application, then you shouldn't use internal storage because when you uninstall the app, all your data will be deleted and other apps will never get that data. For example, if your application is downloading some pdf or storing some image or video that can be used by other applications, then you shouldn't use the internal storage.

#### External Storage

Most of the Android devices have very less internal storage. So, we use some kind of external storage unit to store our data. These storage unit are accessed by everyone i.e. it can be accessed by all the applications in your device. Also, you can access the storage just by connecting the mobile device to some PC.

In order to get access to the external storage, you have to take the ***READ\_EXTERNAL\_STORAGE*** permission from the user. So, any application having this permission can access your application's data.

**Use External Storage**

If the data that is being stored by your application can be used by other application, then you can use external storage. Also, if the file stored by your application is very large as in case of video, then you can store the file in the external storage. If you want the data even after uninstalling the application, then you can use external storage.

If your app is storing some private data that is of no use when the app is not there on the phone, then you should avoid using the external storage.

#### Shared Preferences

If you have a small amount of data to store and you don't want to use the internal storage, then you can use the shared preferences. Shared Preferences are used to store the data in the form of key-value i.e. you will be having one key and based on that key, the corresponding data or value will be stored.

The data stored in the shared preferences will remain with the application until the application is present on your mobile phone. If you uninstall the application, then all the shared preferences will be removed from the device.

**Use Shared Preferences**

When the data that you want to store is very small, then you can use the shared preference in your application. It is recommended not to store more than 100kb of data in shared preferences.

#### Database

Databases are organized collection of data that are stored for future reference. You can store any kind of data in your Database by using some Database Management System. All you need to do is just create the database and do all the operation i.e. insertion, deletion, searching, etc with the help of one query. You will pass the query and the database will return the desired output from the database. SQLite database is one such example of a database in Android.

**Use Database**

When you want to store some structured data, then you can use a database. You can store any kind of data in a database. So, if the data size is very big and you want to access the data in a very easy way, then you can use a database and store your data in a structured format.

#### Shared Storage

Every application in the device has some private storage in the internal memory and you can find this in android/data/your\_package\_name directory. Apart from this internal storage, the rest of the storage is called the Shared Storage i.e. every application with the storage permission can access this part of the memory. This includes media collections and other files of different applications.

In applications that store some data that is being used by some other application, we can use shared storage. But in many cases, other applications only need a small part of the shared storage and rest of the storage is of no use for other applications.

**Use Shared Storage**

When the data stored by your application can be accessed by some other application then you can use shared storage. For example, the images stored by your app can be viewed or edited by some other application present in your phone. Similarly, the pdf stored or generated by your application can be viewed from other applications also. So, in these cases, you can use shared storage.