JÖNKÖPING UNIVERSITY

School of Engineering

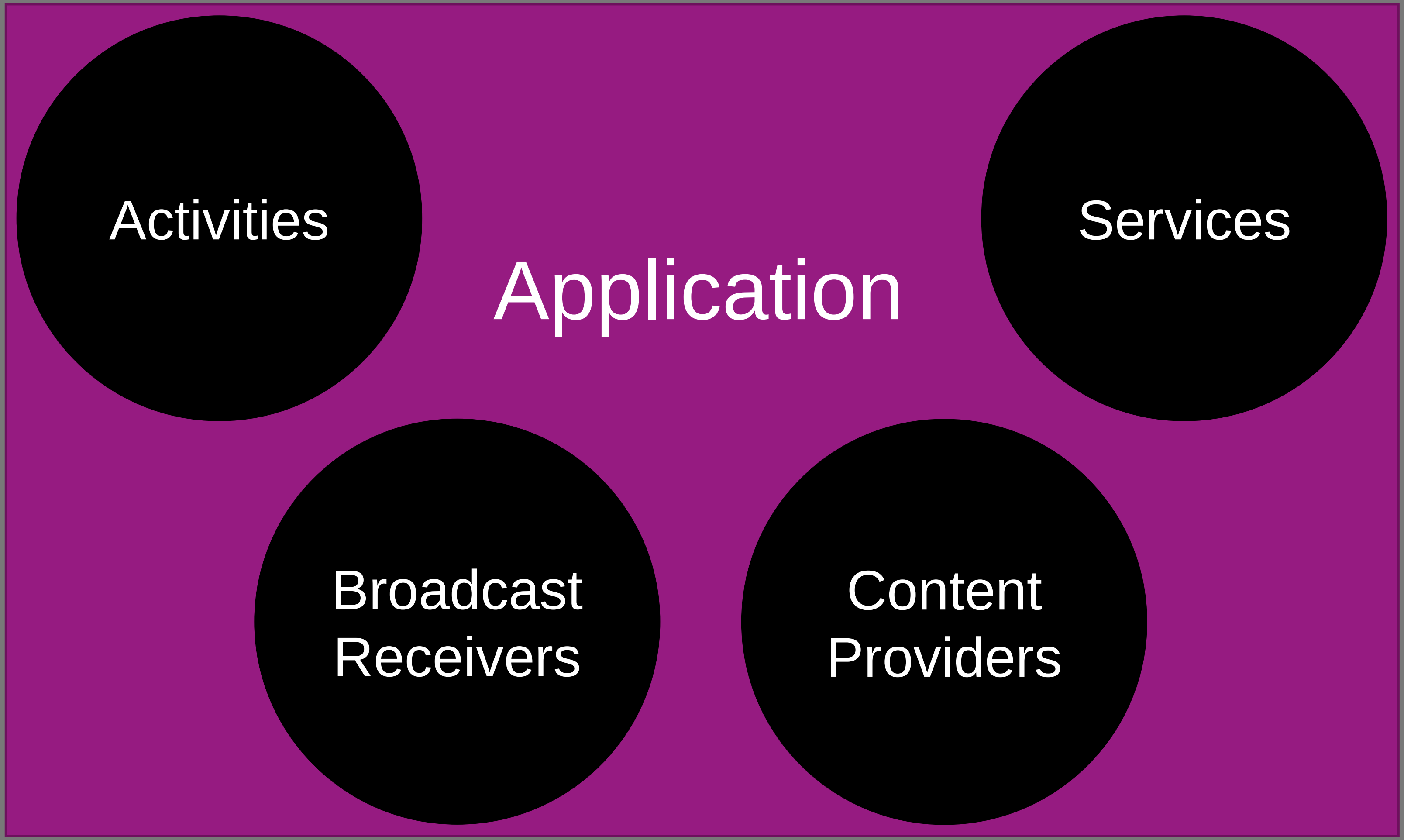
ANDROID CONTENT PROVIDERS

Peter Larsson-Green

Jönköping University

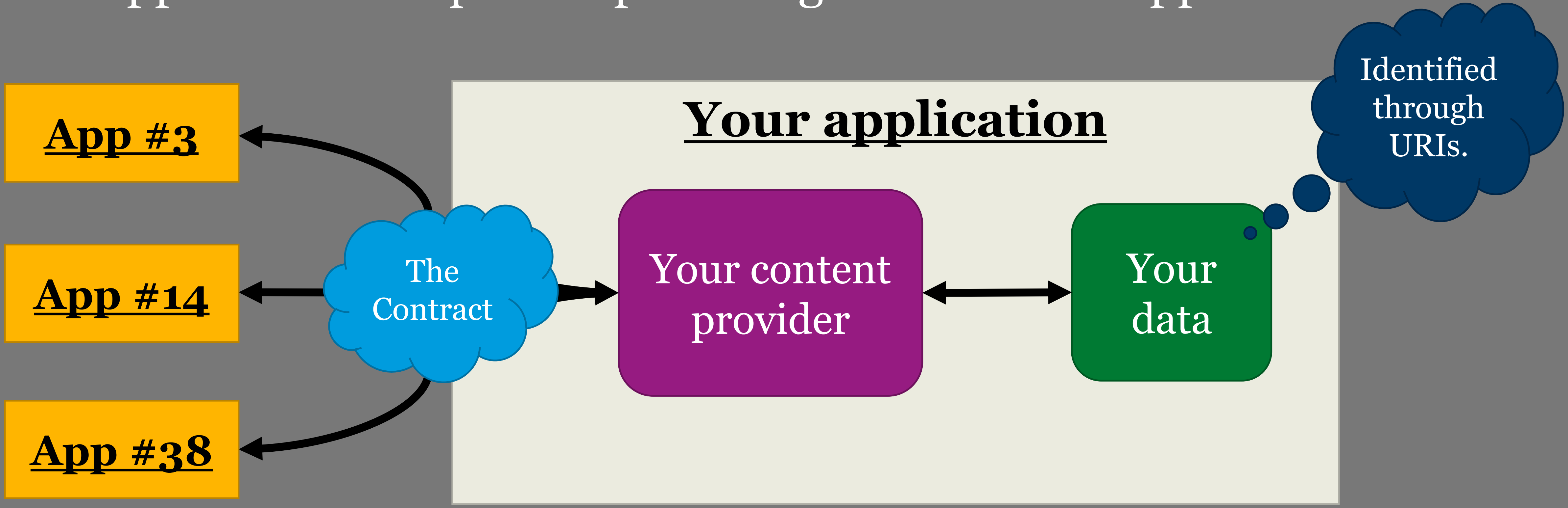
Spring 2021

FUNDAMENTAL APP COMPONENTS



WHAT'S A CONTENT PROVIDER?

An application component providing data to other applications.



WHAT'S A CONTENT PROVIDER?

An application component providing data to other applications.

- In theory, the data can be stored in any way.
- In practice, it is easy to use data from SQLite.

HOW DO I USE A CONTENT PROVIDER?

```
val contentResolver = aContext.contentResolver
```

```
contentResolver.query(theUri, ...)
```

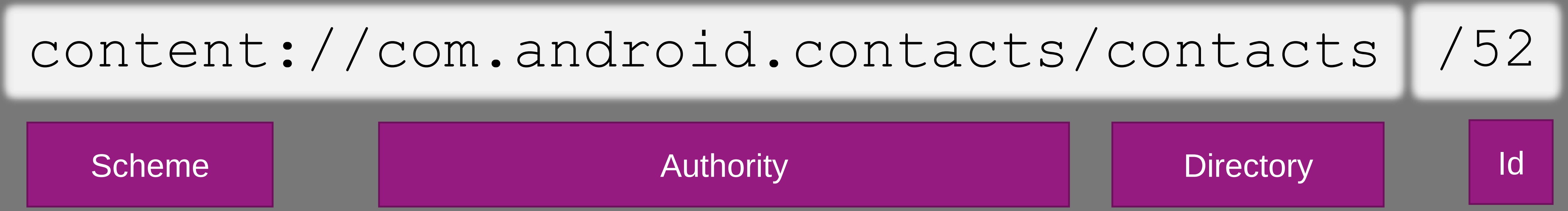
```
contentResolver.insert(theUri, ...)
```

```
contentResolver.update(theUri, ...)
```

```
contentResolver.delete(theUri, ...)
```

THE URI FOR CONTENT PROVIDERS

Identifies data in providers.



```
content://com.android.contacts/contacts /52
```

Useful methods:

```
val uri = Uri.parse("content://authority/collection")
val uri2 = ContentUris.withAppendedId(uri, 37)
val id: Long = ContentUris.parseId(uri2)
```

READING DATA

```
contentResolver.query(
  theUri,
  theProjection,
  theSelection,
  theSelectionArgs,
  sortOrder
)
```

```
contentResolver.query(
  Uri.parse("content://com.android.contacts/contacts"),
  arrayOf("display_name"),
  "display_name = ?",
  arrayOf("Edsger W. Dijkstra"),
  "display_name DESC"
)
```

Don't hardcode the strings, use the contract instead!

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

READING DATA

```
val cursor = contentResolver.query(...)

val count = cursor.getCount()

while(cursor.moveToNext()){
  val aString = cursor.getString(0)
  val anInt = cursor.getInt(1)
}

cursor.close() // (or use the "use" function).
```

INSERTING DATA

```
val values = ContentValues()
values.put("theColumn", theValue)

val uri = contentResolver.insert(
  theUri,
  values
)
```

UPDATING DATA

```kotlin
val values = ContentValues()
values.put("theColumn", theValue)

val numberOfAffectedRows = contentResolver.update(
  theUri,
  values,
  selection,
  selectionArgs
)
```

DELETING DATA

```
val numberOfAffectedRows = contentResolver.delete(
  theUri,
  selection,
  selectionArgs
)
```

CREATING A CONTENT PROVIDER

```
<manifest package="the.package">
  <application ...>
    <provider
      android:name=".MyContentProvider"
      android:authorities="the.package.MyContentProvider"
      android:exported="true"
      android:readPermission="a.permission"
      android:writePermission="a.permission"
    />
  </application>
</manifest>
```

CREATING A CONTENT PROVIDER

```
class MyContentProvider: ContentProvider(){
    override fun onCreate(): Boolean{
        return true
    }
}
```

Did everything go well?

CREATING A CONTENT PROVIDER

```
class MyContentProvider: ContentProvider(){
  override fun query(uri: Uri, projection: Array<String>, selection: String,
                     selectionArgs: Array<String, sortOrder: String): Cursor{}
  override fun insert(uri: Uri, values: ContentValues): Uri{}
  override fun delete(uri: Uri, selection: String, selectionArgs: Array<String>):
                                                                         Int{}
  override fun update(uri: Uri, values: ContentValues, selection: String,
                                  selectionArgs: Array<String>): Int{}
}
```

These methods must be thread safe!

PROVIDING FILES

Content providers can also provide read and write streams to files.

```
val contentResolver = aContext.contentResolver
```

```
val is = contentResolver.openInputStream(theUri)
```

```
val os = contentResolver.openOutputStream(theUri, "rw")
```

"w","wa",
"rw", "rwt"

In your content provider, override:

```
openFile(Uri uri, String mode)
```

ADDING A FILE PROVIDER

```
<manifest package="the.package">
  <application ...>
      <provider
        android:name="androidx.core.content.FileProvider"
        android:authorities="se.ju.larpet.fileprovider"
        android:exported="false"
        android:grantUriPermissions="true">
          <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/file_provider_paths" />
      </provider>
  </application>
</manifest>
```

ADDING A FILE PROVIDER

```
<manifest package="the.package">
  <application ...>
      <provider
        android:name="androidx.core.content.FileProvider"
        android:autho
        android:expor
        android:grant
          <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:r
      </provider>
  </application>
</manifest>
```

Part of URI other apps see.

Actual sub directory.

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
  <files-path name="pics" path="cars/" />
</paths>
```

```
/our-app-internal-folder/cars/file.txt
```

```
content://.../pics/file.txt
```

EXAMPLE: TAKING PICTURE

```
val file = File(aContext.filesDir, "cars/my-file.jpeg")
file.parentFile.mkdirs()
file.createNewFile()
val fileUri = FileProvider.getUriForFile(
  aContext,
  "se.ju.larpet.fileprovider",
  file
)
val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri)
intent.flags = Intent.FLAG_GRANT_WRITE_URI_PERMISSION
aContext.startActivityForResult(intent, 1234)
```

LISTENING FOR DATA CHANGES

```
ContentObserver yourContentObserver = new ContentObserver(){
  public ContentObserver(){ super(new Handler()); }
  public void onChange(boolean selfChange){ }          /* API L <= 15 */
  public void onChange(boolean selfChange, Uri uri){ } /* 16 <= API L */
};
contentResolver.registerContentObserver(
  theUri,
  false,
  yourContentObserver
);
```

`false` = exact URI.
`true` = exact URI + children

```
contentResolver.unregisterContentObserver(yourContentObserver);
```

THE URI MATCHER

Zero or more digits.

```
val matcher = UriMatcher(UriMatcher.NO_MATCH)
matcher.addURI("the.authority", "the/path", 1)
matcher.addURI("the.authority", "the/path-2", 2)
matcher.addURI("the.authority", "the/path/#", 3)
matcher.addURI("the.authority", "the/path-2/*", 4)

val uri = Uri.parse("content://the.authority/the/path-2")
val two = matcher.match(uri)
```

Zero or more characters.

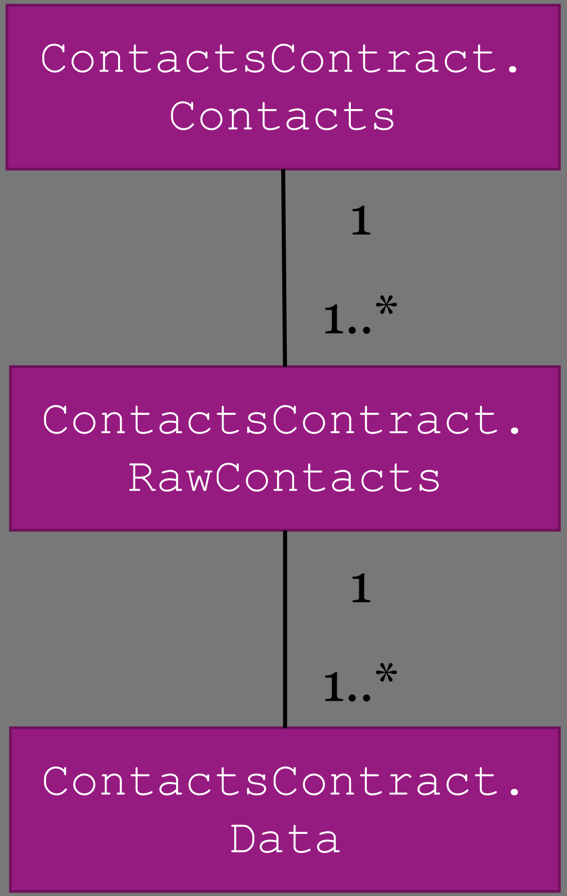
CREATING A CONTENT PROVIDER

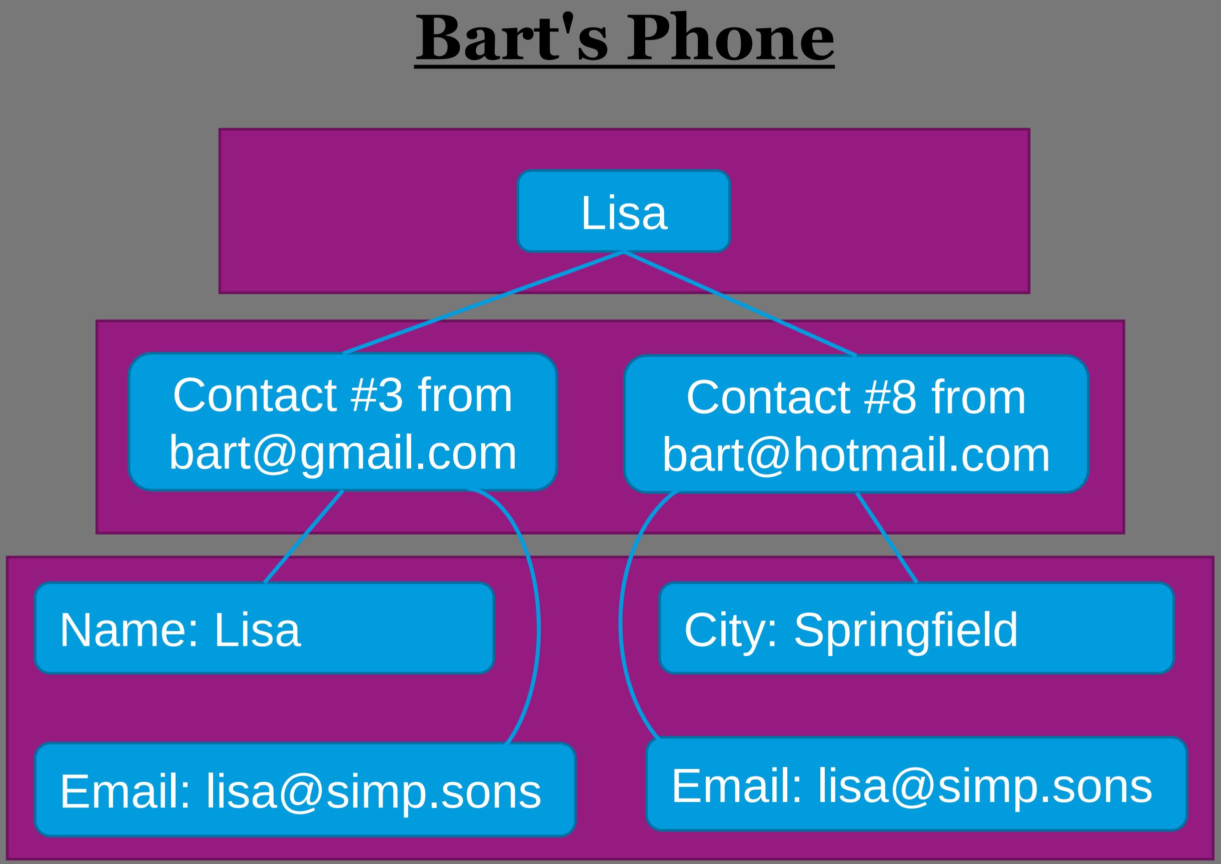
```
class MyContentProvider: ContentProvider(){
   override fun getType(uri: Uri): String{
      if(/* uri points to collection */){
        return "vnd.android.cursor.dir/vnd.package.name";
      }else{
        return "vnd.android.cursor.item/vnd.package.name";
      }
   }
```

ContentResolver.
CURSOR_DIR_BASE_TYPE

ContentResolver.
CURSOR_ITEM_BASE_TYPE

HOW CONTACTS ARE ORGANIZED





CONTACT PROVIDER'S CONTRACT

```
ContactsContract.Contacts.CONTENT_URI
ContactsContract.Contacts._ID,
ContactsContract.Contacts.DISPLAY_NAME
```

```
ContactsContract.CommonDataKinds.Phone.CONTENT_URI
ContactsContract.CommonDataKinds.Phone.NUMBER
ContactsContract.CommonDataKinds.Phone.CONTACT_ID
```

```
ContactsContract.CommonDataKinds.Email.CONTENT_URI
ContactsContract.CommonDataKinds.Email.ADDRESS
ContactsContract.CommonDataKinds.Email.CONTACT_ID
```

PATTERN FOR NOTIFYING CHANGES

Use content providers to notify changes.

- Need to properly implement `query`, `insert`, `update` & `delete`.
- To work properly, data may only be changed through these methods on the content provider.
    - In fragments/activities, work with the data through the content provider.

NOTIFYING CHANGES

```
public class MyContentProvider extends ContentPrivider{
  public Uri insert(Uri uri, ContentValues values){
    // Do the insertion...
    getContext().getContentResolver().notifyChange(
      theUri,
      theContentObserver
    );
  }
}
```

In many cases `null`.