EMBEDDED SYSTEMS PROGRAMMING 2016-17

Android Content Providers

APP COMPONENTS

- **Activity**: a single screen with a user interface

- **Broadcast receiver**: responds to system-wide broadcast events. No user interface

- **Service**: performs (in the background) long-running operations (e.g., music playback). No user interface

- **Content provider**

CONTENT PROVIDER

- Encapsulates structured data that need to be shared across applications

- The **official way to share data** across applications

- Encapsulates data = provides a common interface for adding, modifying, querying, deleting,... data

- Does not provide a way to store data. “How a content provider actually stores its data under the covers is up to its designer”

DATA MODEL

- Data are exposed as **tables** (like in a database)

- Multiple tables can be handled by a single content provider

- Every table includes an **`_ID` column**,
  which holds a unique numeric ID for each record

CLASSES (1/2)

`android.content` package:

- **`ContentProvider`** abstract class
  Encapsulates data

- **`ContentResolver`** abstract class
  Provides access to data stored in a `ContentProvider`

`android.net` package:

- **`Uri`** class
  Provides a mean of identifying tables and rows

URI

- **U**niform **R**esource **I**dentifier
- Described by RFC 2396

- Syntax:
  - **Scheme name** (`content` for content providers)
  - “`://`”
  - **Authority** (in this case, the name of the content provider)
  - **Path to resource**. Path components are separated by “/”

URI IN CONTENT PROVIDERS: EXAMPLES

- All URIs for providers begin with `content://`
- URI of a content provider:

```
content://it.unipd.dei.esp1112.AddressBookProvider
```

- URI of a table (the table named `People`):

```
content://it.unipd.dei.esp1112.AddressBookProvider/People/
```

- URI of a single row (the row with `_ID`=24):

```
content://it.unipd.dei.esp1112.AddressBookProvider/People/24
```

CLASSES (2/2)

- Content providers must be declared in `AndroidManifest.xml`

- The system, not the user, instantiates content providers (i.e., objects from subclasses of `ContentProvider`)

- One instance for each class, hence one instance for each type of content provider

- The instance can deal with multiple requests by communicating with multiple content resolvers

EXAMPLES (1/2)

- Declaring a content provider

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unipd.dei.esp1112.cp1"
    android:versionCode="1"
    android:versionName="1.0.0">

    <application android:icon="@drawable/icon" android:label="@string/app_name">

     <provider android:name="MyAddressBook"
               android:authorities="it.unipd.dei.esp1112.AddressBookProvider"
               android:exported="true">
        <grant-uri-permission android:pathPattern=".*" />
     </provider>
...
    </application>

</manifest>
```

EXAMPLES (2/2)

- Asking for the permission to read data from Android's `Contacts` content provider

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unipd.dei.esp1112.cp2"
    android:versionCode="1"
    android:versionName="1.0.0">

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

    <application android:icon="@drawable/icon" android:label="@string/app_name">
...
    </application>

</manifest>
```

CONTENTPROVIDER CLASS: METHODS (1/3)

- **`boolean onCreate()`**
  Initializes the content provider.
  Advice: lengthy initializations, such as opening an `SQLiteDatabase`, should be deferred until the content provider is actually used

- **`String getType(Uri uri)`**
  Returns the MIME type of data in the content provider at the given URI

CONTENTPROVIDER CLASS: METHODS (2/3)

- **`Uri insert(Uri uri, ContentValues values)`**
  Inserts a new row in the table at the URI `uri`.
  Returns the URI for the newly inserted item

- **`int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)`**
  Updates all rows matching the `selection` filter in the table (or record) identified by `uri`. New values are contained in `values`, which is a mapping from column names to new column values.
  Returns the number of affected rows

- **`int delete(Uri uri, String selection, String[] selectionArgs)`**
  Deletes all rows matching the `selection` filter in the table (or record) identified by `uri`.
  Returns the number of affected rows

CONTENTPROVIDER CLASS: METHODS (3/3)

- **`Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`**

  Performs a query among all rows matching the `selection` filter in the table (or record) identified by `uri`. `projection` contains a list of columns to put into the result.

  The result of the query is returned as a `Cursor` object

CONTENTRESOLVER CLASS: METHODS

- All data modifications and queries are performed indirectly via `ContentResolver` methods

- `ContentResolver`s are not instantiated, but obtained by invoking **`getContentResolver()`** from within an activity or other application component

- Same methods of a `ContentProvider`: `insert()`, `update()`, `delete()`, `query()`, …

EXAMPLES (1/2)

- Obtaining a single record from Android's addressbook

```
import android.provider.Contacts.People;
import android.net.Uri;
import android.database.Cursor;
import android.content.ContentResolver;

...

    // Use the Uri class to build the URI
    Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

    // Obtain a content resolver
    ContentResolver cr = getContentResolver();

    // Query for the specific record
    Cursor cur = cr.query(myPerson, null, null, null, null);

...
```

EXAMPLES (2/2)

- Obtaining a set of records from Android's addressbook

```
import android.provider.Contacts.People;
import android.net.Uri;
import android.database.Cursor;
import android.content.ContentResolver;

...

   // Form an array specifying which columns to return.
   // The names of the columns are available as constants in the People class
   String[] projection = new String[] {People._ID, People._COUNT,
                                       People.NAME, People.NUMBER};

   // Get the base URI for the People table in the Contacts content provider
   Uri contacts =  People.CONTENT_URI;

   // Make the query
   Cursor Cur = getContentResolver().query
                    (contacts,
                     projection, // Which columns to return
                     null,       // Which rows to return (all rows)
                     null,       // Selection arguments (none)
                     People.NAME + " ASC"); // Put results in ascending order by name
...
```

REFERENCES

- Content Providers
- "Working with Content Providers" tutorial on Tuts+
- RFC 2396

LAST MODIFIED: MAY 2, 2017