CS 5450

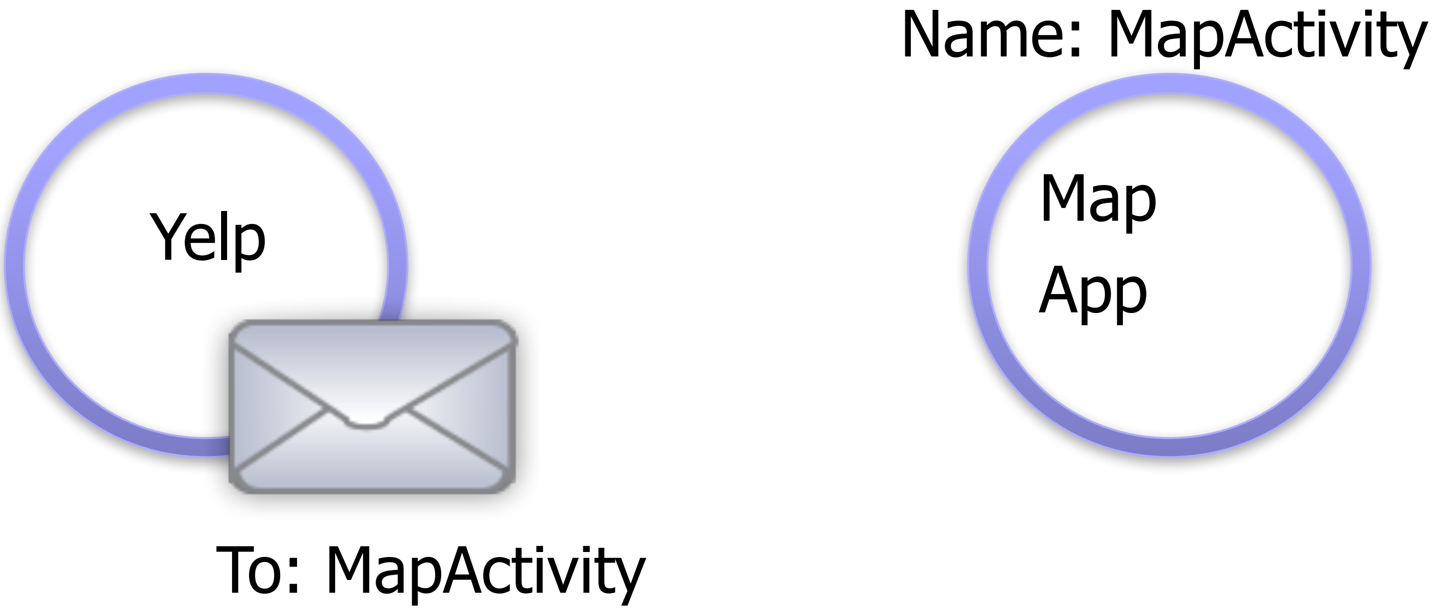
Android

Vitaly Shmatikov

Structure of Android Applications

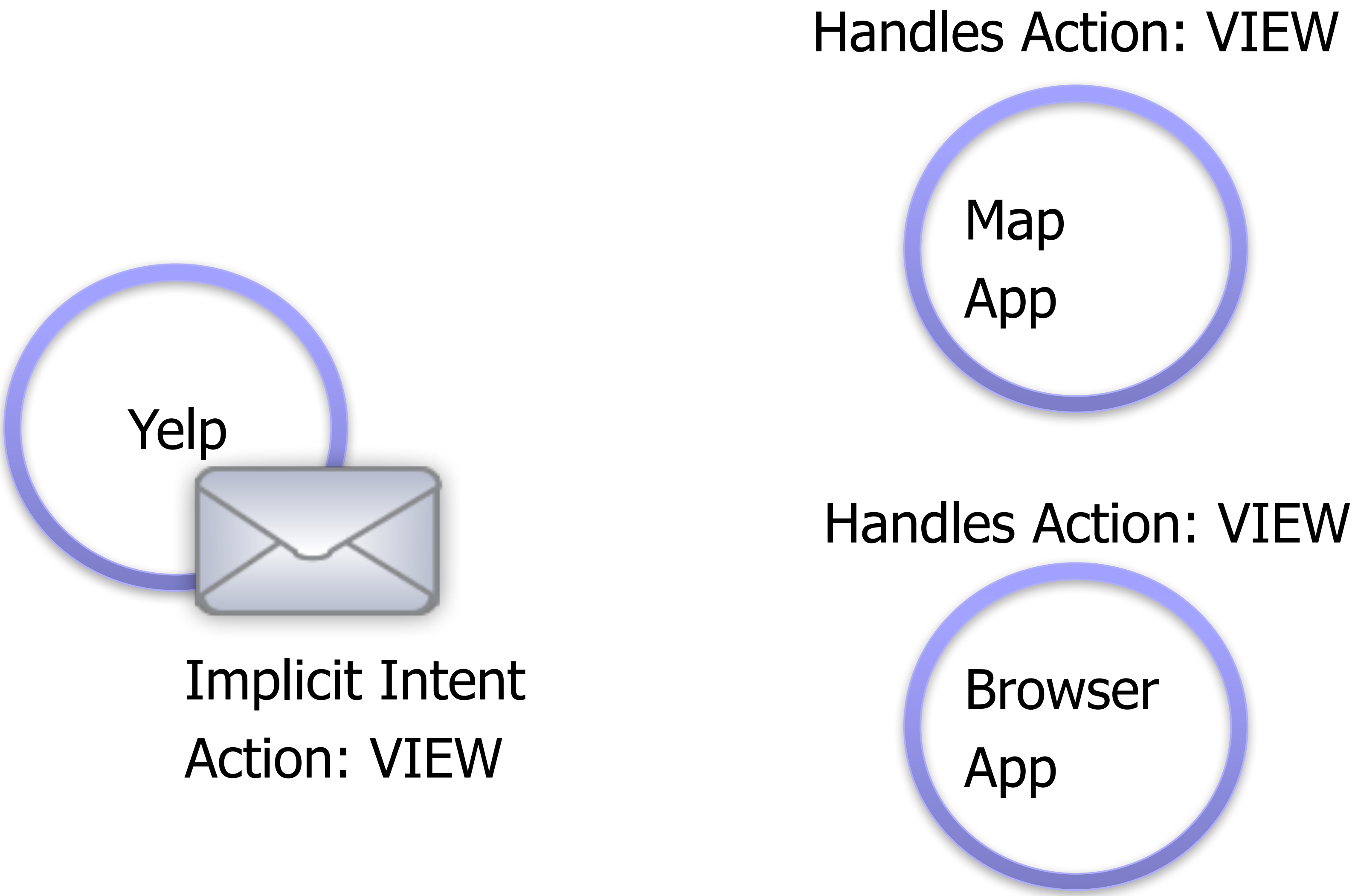
- Applications include multiple components
  - Activities: user interface
  - Services: background processing
  - Content providers: data storage
  - Broadcast receivers for messages from other apps
- Intent: primary messaging mechanism for interaction between components

Explicit Intents



Only the specified destination receives this message

Implicit Intents



Handles Action: VIEW

Map
App

Yelp

Handles Action: VIEW

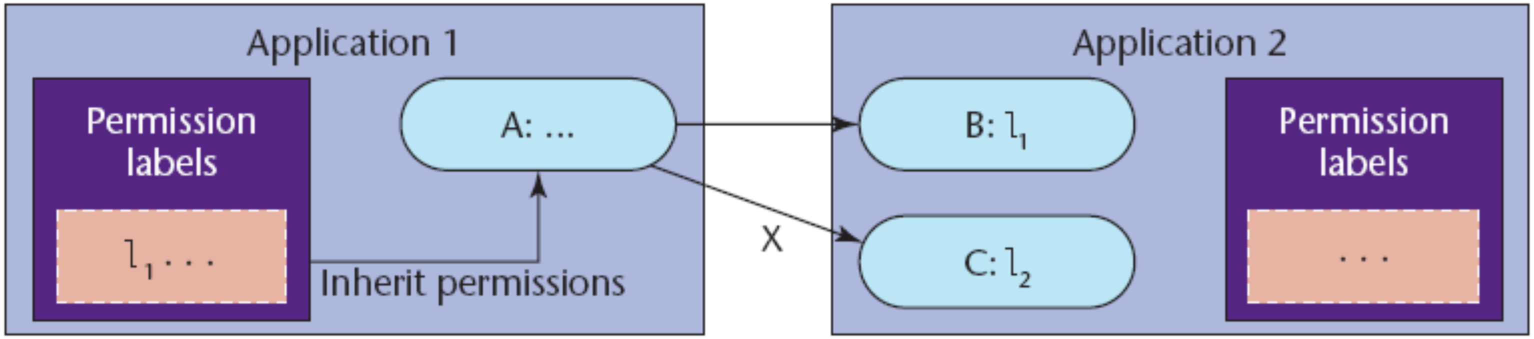
Implicit Intent
Action: VIEW

Browser
App

Android Security Model

Access permitted if labels assigned to the invoked component are in the collection of invoking component

- Based on permission labels assigned to applications and components



- Every app runs as a separate user
  - Underlying Unix OS provides system-level isolation
- Reference monitor in Android middleware mediates inter-component communication

Mandatory Access Control

- Permission labels are set (via manifest) when app is installed and cannot be changed
- Permission labels only restrict access to components, they do not control information flow – means what?
- Apps may contain "private" components that should never be accessed by another app (example?)
- If a public component doesn't have explicit permissions listed, it can be accessed by any app

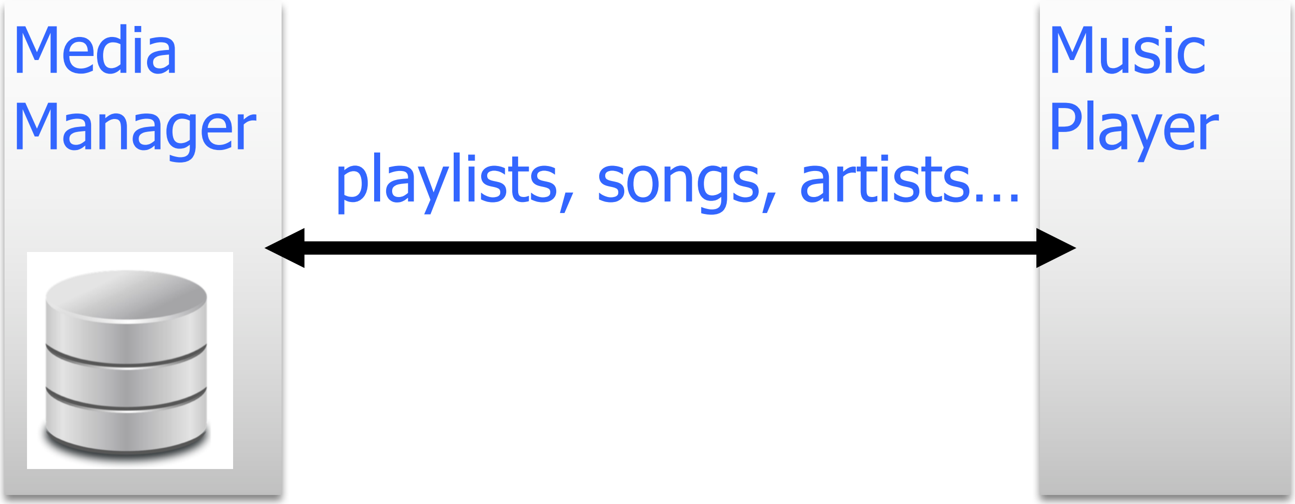
System API Access

- System functionality (eg, camera, networking) is accessed via Android API, not system components
- App must declare the corresponding permission label in its manifest + user must approve at the time of app installation
- Signature permissions are used to restrict access only to certain developers
  - Ex: Only Google apps can directly use telephony API

Refinements

- Permission labels on broadcast intents
  - Prevents unauthorized apps from receiving these intents – why is this important?
- Pending intents
  - Instead of directly performing an action via intent, create an object that can be passed to another app, thus enabling it to execute the action
  - Invocation involves RPC to the original app
  - Introduces delegation into Android's MAC system

Using Media Data

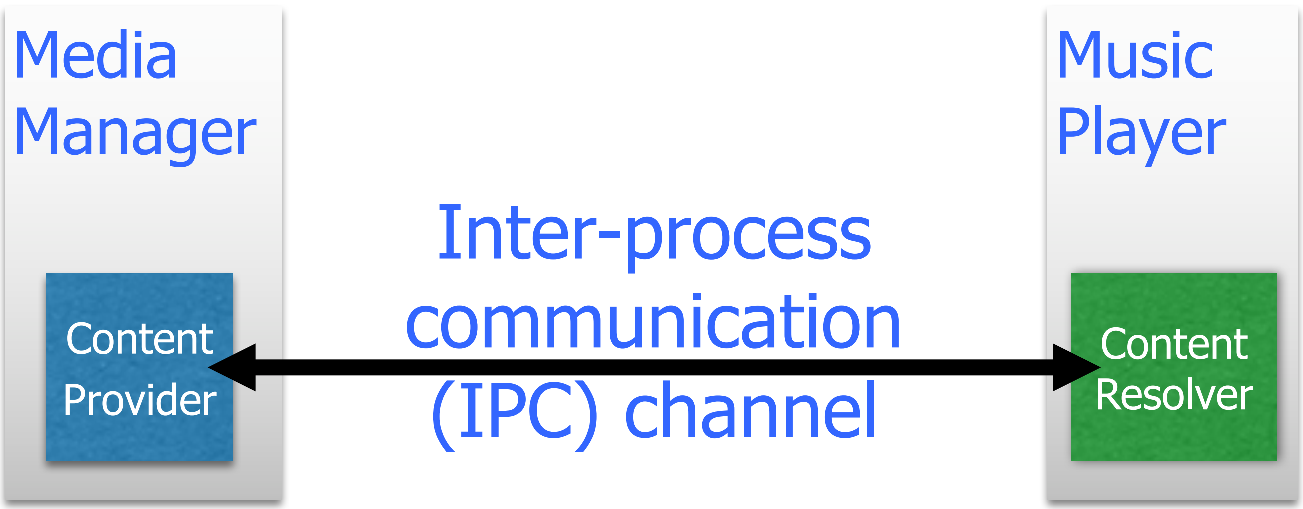


Media Manager

playlists, songs, artists…

Music Player

Using Media Data



Media Manager

Content Provider

Inter-process communication (IPC) channel

Music Player

Content Resolver

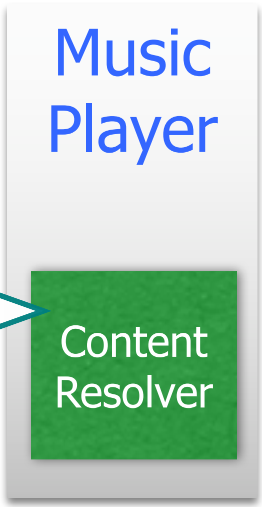
Client Side: Content Resolver

Implemented by Android:

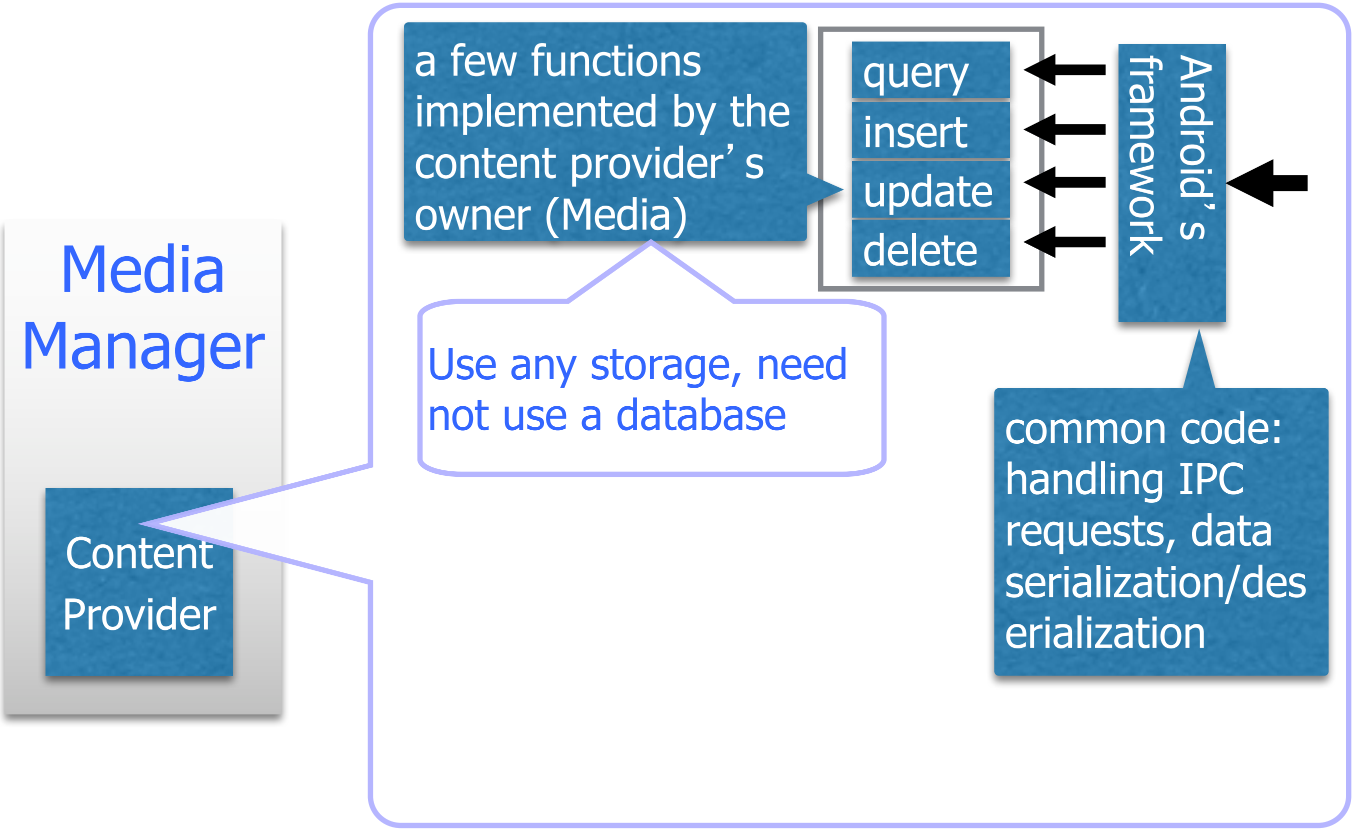
getContentResolver()

API: “CRUD” — similar to database

- insert (**C**reate)
- query (**R**etrieve)
- **U**pdate
- **D**elete



Service Side: Content Provider



Media Manager

Content Provider

a few functions implemented by the content provider’s owner (Media)

query

insert

update

delete

Android’s framework

Use any storage, need not use a database

common code: handling IPC requests, data serialization/deserialization

Built-In Content Providers

- Contacts
- Media
- Calendar
- User dictionary

…

Example: Built-In User Dictionary

- Stores the spellings of non-standard words that the user wants to keep
- Backed by a database 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

Query from Another App

Get the content resolver object

```
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
```

URI: an identifier to locate data in the user dictionary

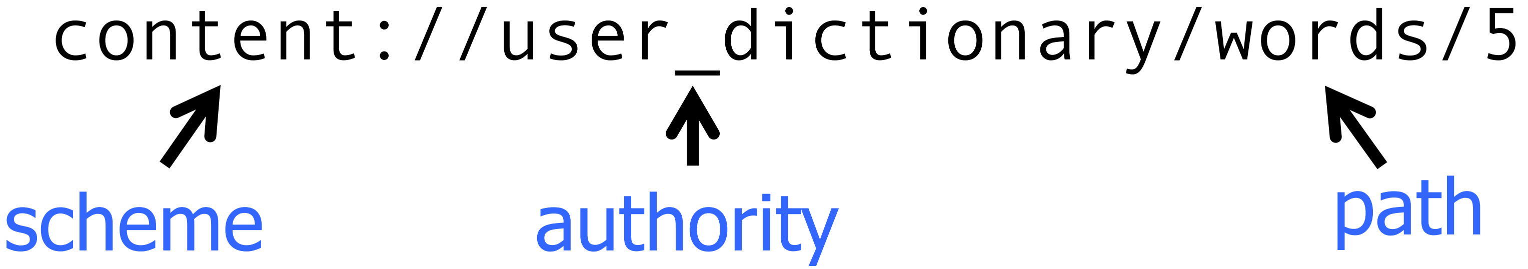
Content URIs

- Scheme: always "content"
- Authority: name of entire provider

used by Android to identify a content provider

- Path (optional):
  - Data type path
  - Instance identifier

used by the content provider to identify **internal** objects



```
content://user_dictionary/words/5
```

scheme

authority

path

For non-built-in apps: com.example.<appname>.provider

Why Create a Content Provider?

- Want to offer complex data or files to other apps
- Want to allow users to copy complex data from your app into other apps
- Want to provide custom search suggestions using the search framework

Creating a Content Provider

- Design URI-to-data mapping
- Manifest declaration
- Implementation
- Permissions

URI-to-data Mapping

- authority: user_dictionary
- path:
  - /words: all words
  - /words/<id>: a specific word
- Use UriMatcher

```
sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
sUriMatcher.addURI(AUTHORITY, "words", WORDS);
sUriMatcher.addURI(AUTHORITY, "words/#", WORD_ID);
```

Declare in Manifest

- A content provider is an app component

```
<application>
…
     <!-- The Content Provider is declared -->
     <provider android:name="UserDictionaryProvider"
          android:authorities=“user_dictionary"
          android:syncable="false"
          android:multiprocess="false"
          android:exported="true"

   android:readPermission="android.permission.READ_USER_DICTIONARY"

   android:writePermission="android.permission.WRITE_USER_DICTIONARY"
   />
</application>
```

Implementation

```
public class UserDictionaryProvider extends
ContentProvider {
   insert(…);
   query(…);
   update(…);
   delete(…);
   …
}
```

Implementing “query”

```
public Cursor query(
        Uri uri,
        String[] projection,
        String selection,
        String[] selectionArgs,
        String sortOrder);
```

Match URI

```
switch (sUriMatcher.match(uri)) {
case WORDS:
    qb.setTables(USERDICT_TABLE_NAME);
    qb.setProjectionMap(sDictProjectionMap);
    break;
case WORD_ID:
    qb.setTables(USERDICT
    qb.setProjectionMap(s
    qb.appendWhere(
        "_id" + "=" + uri.getPathSegments().get(1));
    break;
default:
    throw new IllegalArgumentException(
        "Unknown URI " + uri);
}
```

content://user_dictionary/words/1

path segments: [“words”, “1”]

Query DB, Return Cursor

```
// If no sort order is specified use the default
String orderBy;
if (TextUtils.isEmpty(sortOrder)) {
    orderBy = Words.DEFAULT_SORT_ORDER;
} else {
    orderBy = sortOrder;
}

// Get the database and run the query
SQLiteDatabase db = mOpenHelper.getReadableDatabase();
Cursor
      s

// Tell                                              en its source
data changes
c.setNotificationUri(
     getContext().getContentResolver(), uri);

return c;
```

Register a ContentObserver

Allow Android’s “CursorLoader” mechanism to automatically re-fetch data

Implementing Insert

```
@Override
public Uri insert(Uri uri, ContentValues initialValues) {
    // Validate the requested uri
    if (sUriMatcher.match(uri) != WORDS) {
        throw new IllegalArgumentException("Unknown URI " + uri);
    }
    ContentValues values;
  … // sanitize initialValues and store to values

    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    long rowId = db.insert(
       USERDICT_TABLE
    if (rowId > 0) {
        Uri wordUri = ContentUris.withAppendedId(
                 UserDictionary.Words.CONTENT_URI, rowId);
        getContext().getContentResolver().notifyChange(
                 wordUri, null);
        mBackupManager.dataChanged();
        return wordUri;
    }
    throw new SQLException("Failed to insert row into " + uri);
}
```

Return the inserted URIs

Notify content observers

Permissions in Manifest

```
</application>
  …
    <!-- The Content Provider is declared -->
    <provider android:name="UserDictionaryProvider"
      android:aut
      android:syr
      android:multiprocess  alse"
      android:exported="true"
      android:readPermission="android.permission.READ_USER_DICTIONARY”
     android:writePermission="android.permission.WRITE_USER_DICTIONARY" />
</application>
```

Enable sharing with other apps

Read and write permissions

Provider-Level Permissions

- Single read-write provider-level permission
  - 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 permissions
  - Specify them with the android:readPermission and android:writePermission attributes of the <provider> element
  - They take precedence over the permission required by android:permission

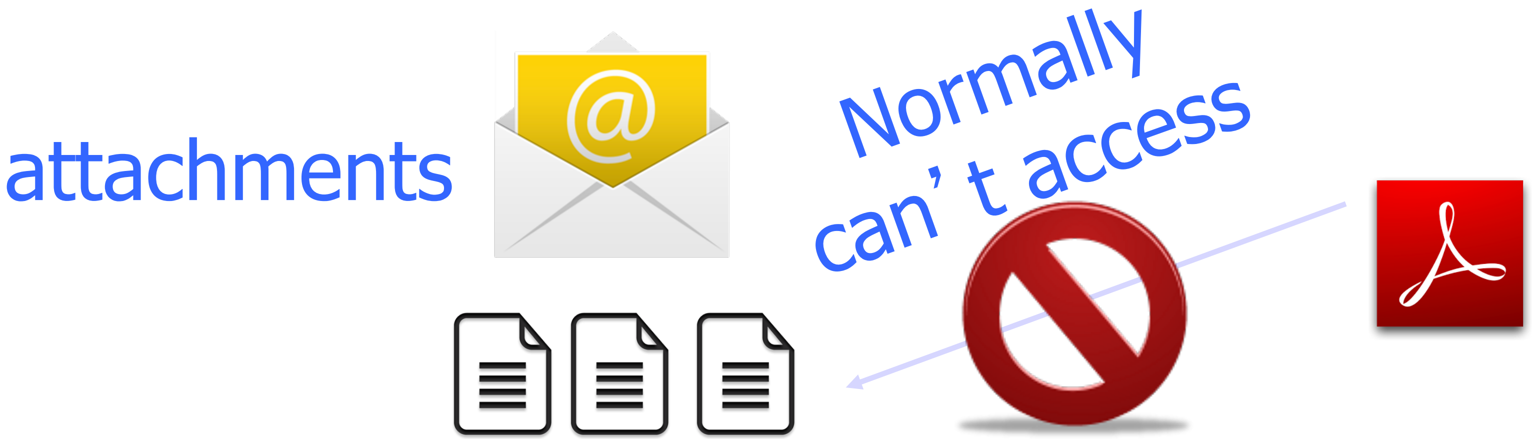
Path-Level Permissions

- Specify each URI with a <path-permission> child element of the <provider> element
  - For each content URI, can specify a read/write permission, a read permission, a write permission, or all three.
- Path-level permission takes precedence over provider-level permissions

Temporary Permissions

- Temporarily grant an app access in the context of an invocation using an intent, to a specific URI specified in the intent
  - Revoked when this invocation ends

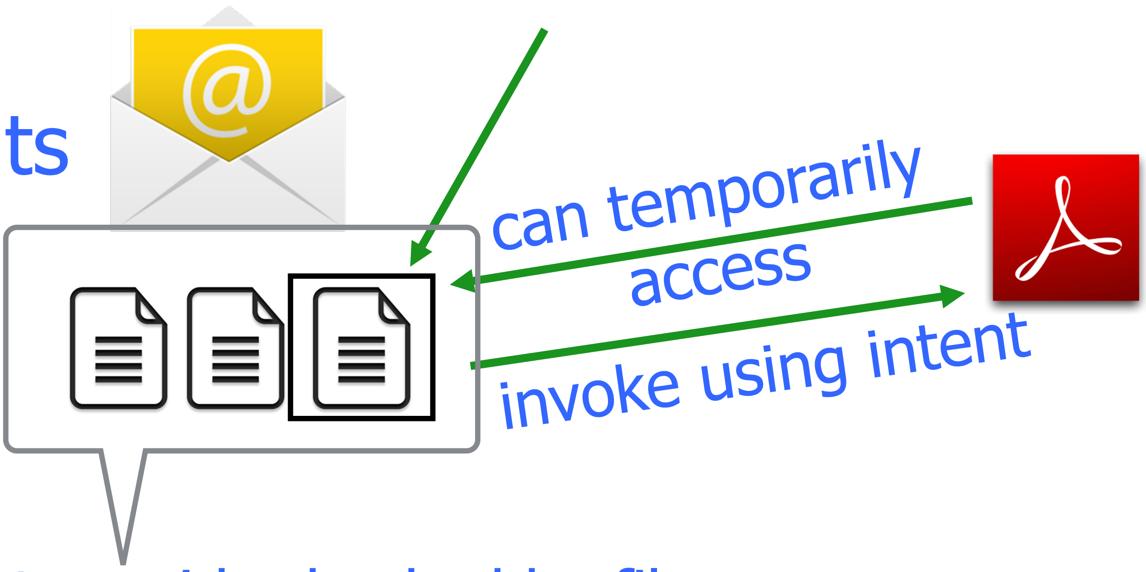
Example: Email Attachments



attachments

Normally can’ t access

Example: Email Attachments



attachments

user clicks

can temporarily access

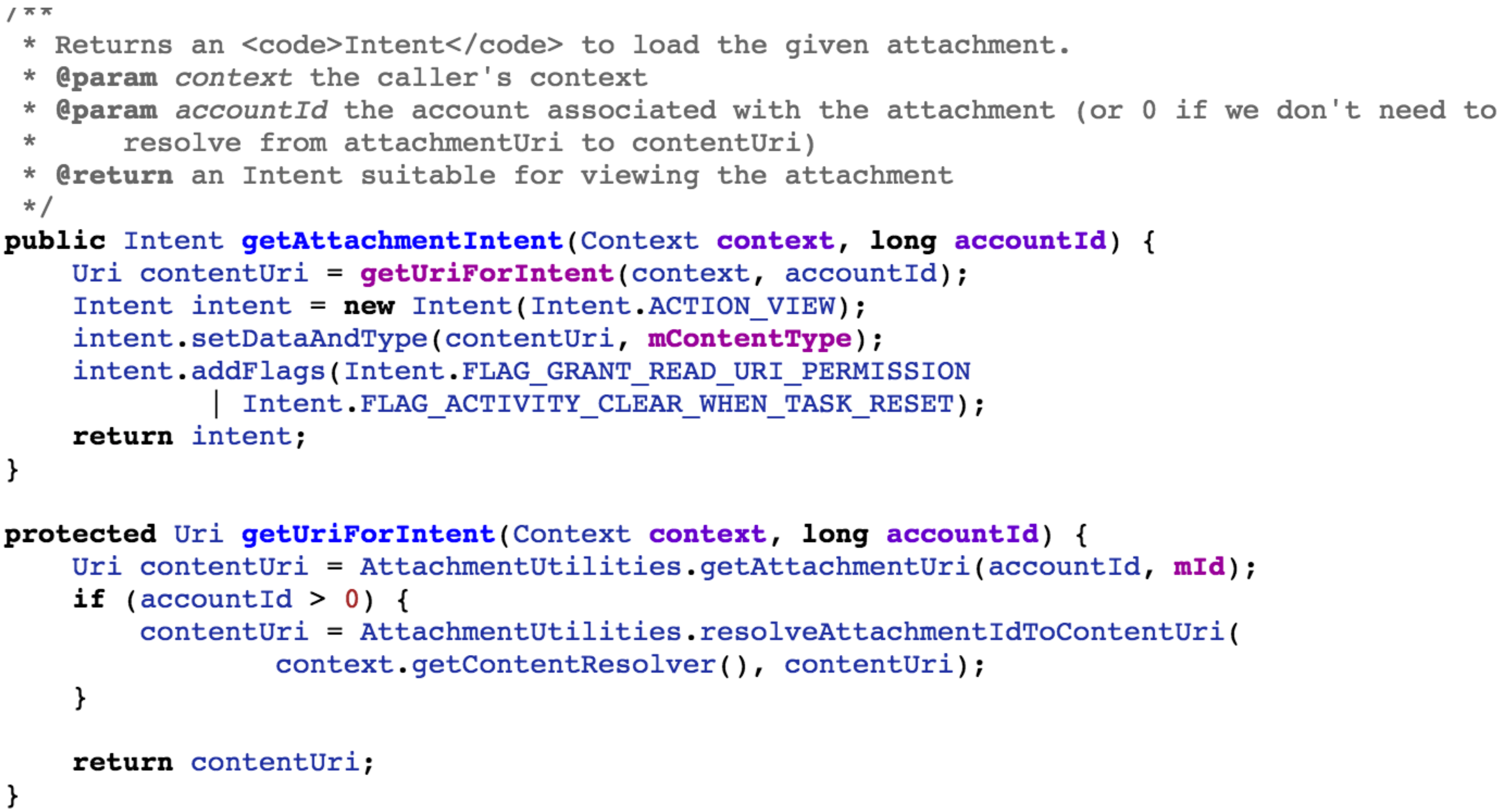
invoke using intent

a content provider backed by files

Temporary Permissions

- Manifest: assert android:grantUriPermissions attribute in the <provider> element
  - The scope of these permissions can be further limited by the <grant-uri-permission>
- Intent (runtime): using the FLAG_GRANT_READ_URI_PERMISSION and FLAG_GRANT_WRITE_URI_PERMISSION flags in the Intent object that activates the component

Invoke Using Intent

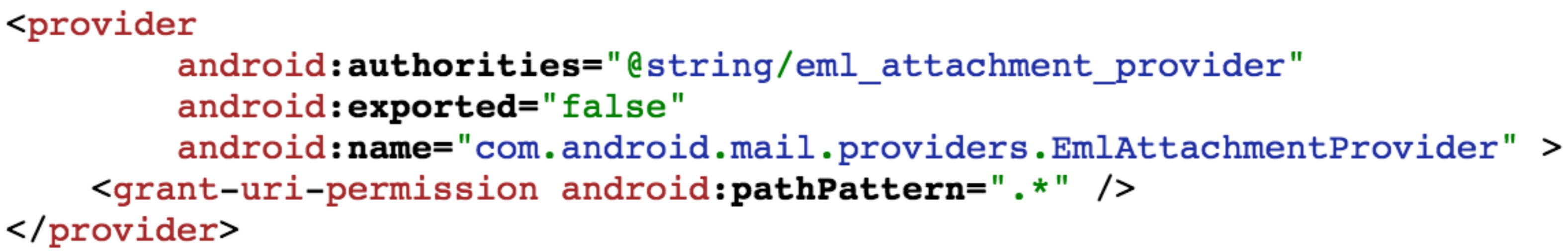


```
/**
 * Returns an <code>Intent</code> to load the given attachment.
 * @param context the caller's context
 * @param accountId the account associated with the attachment (or 0 if we don't need to
 *     resolve from attachmentUri to contentUri)
 * @return an Intent suitable for viewing the attachment
 */
public Intent getAttachmentIntent(Context context, long accountId) {
    Uri contentUri = getUriForIntent(context, accountId);
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setDataAndType(contentUri, mContentType);
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION
            | Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
    return intent;
}

protected Uri getUriForIntent(Context context, long accountId) {
    Uri contentUri = AttachmentUtilities.getAttachmentUri(accountId, mId);
    if (accountId > 0) {
        contentUri = AttachmentUtilities.resolveAttachmentIdToContentUri(
                context.getContentResolver(), contentUri);
    }

    return contentUri;
}
```

Enable in Manifest



```
<provider
        android:authorities="@string/eml_attachment_provider"
        android:exported="false"
        android:name="com.android.mail.providers.EmlAttachmentProvider" >
    <grant-uri-permission android:pathPattern=".*" />
</provider>
```

Use Files in Content Provider

```
public ParcelFileDescriptor openFile(
    Uri uri, String mode)
  throws FileNotFoundException
```

- FileProvider: a subclass of ContentProvider
  - Implemented by Android
  - Supports simple filename-to-URI mapping