Android System Intent

Objectives

- Understanding Intent Architecture
- Sending/Launching Intents
  - Intent without result
  - Intent with result
- Receiving Intents
  - Role of Intent Filters
- There is no equivalent builtin feature in iOS

On GitHub: android-intent-compose

Android Intent Class (Android **Messaging** Framework)

- Related to the Command design pattern in software engineering
  - The Command pattern encapsulates operations as an object
  - The Android Intent encapsulates a **message** to perform an operation
- Main data fields of the Intent class
  - **action**: the generate action to be performed
  - **data**: the data to operate on, expressed as URI (Uniform Resource Identifier)
  - **category**: classification identification
  - **type**: MIME type of the intent data such as text/plain, image/jpg, image/png, etc.
  - **component**: fully qualified package name of the intended recipient
  - **extra**: a “dictionary”/”map” for passing additional information (besides data above)

Intent Usage

- Main use: request an action from **another component,** which can be
  - Another activity within the same app
  - A service within the same app
  - An activity in a different app
  - Any activity available on the device
- Types of Intent
  - Explicit: the target component is fully specified (package name + activity name)
  - Implicit: only the generic action is provided in the message
- Broadcast Receiver & Intent Filters

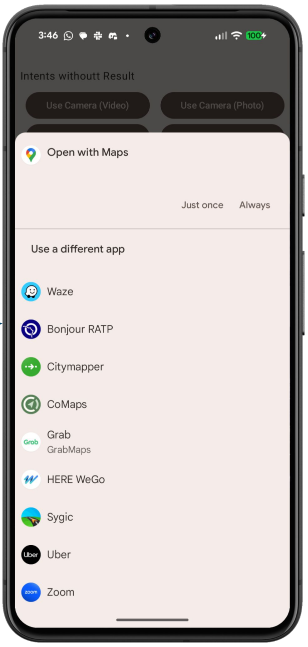
List of Common Intents

- Camera: `Intent.ACTION_(IMAGE|VIDEO)_CAPTURE`
- Contact
  - Select: Intent.ACTION_PICK
  - View: Intent.ACTION_VIEW
  - Edit: Intent.ACTION_EDIT
  - Add: Intent.ACTION_INSERT
- Email: Intent.ACTION_SEND, Intent.ACTION_SEND_MULTIPLE
- Document: Intent.ACTION_OPEN_DOCUMENT
- Taxi: ReserveIntents.ACTION_TAXI_RESERVATION
- Map: Intent.ACTION_VIEW
- Phone Call: Intent.ACTION_DIAL, Intent.ACTION_CALL
- Settings: Intent.ACTION_WIFI_SETTING, ACTION_BLUETOOTH_SETTINGS, ...

Intent Resolution

Intent Type	Description
Implicit	Only the generic action is specified in the message. The actual recipient of the message is resolved at runtime. If **multiple apps** are capable of handling the message, a dialog will pop up
Explicit	The target component is fully specified (package name + activity name). If the target app does not exist, it triggers a runtime error

multiple apps can handle the ***implicit Intent***
ACTION_VIEW geo:42.96,-85.89



Launching an Intent requires the Android runtime to manage the UI from two or more applications. Hence, the launch is typically triggered from your View (and not in ViewModel)

Using Implicit Intent

```
Button(onClick = {
   // Map View at a specified Latitude & Longitude
   val mapIntent = Intent(Intent.ACTION_VIEW).apply {
       data = Uri.parse("geo:42.964577, -85.891341")
   }
   context.startActivity(mapIntent)
}) {
   Text("Show Map")
}
```

Using Explicit Intent

```
Button(onClick = {
    // Google Map at a specified Latitude & Longitude
    val mapIntent = Intent(Intent.ACTION_VIEW).apply {
        data = Uri.parse("geo:42.964577, -85.891341")
       `package` = "com.google.android.apps.maps"
    }
    context.startActivity(mapIntent)
}) {
    Text("Show Map")
}
```

backtick around package *is required by Kotlin syntax*

Adding Permissions in AndroidManifest.xml

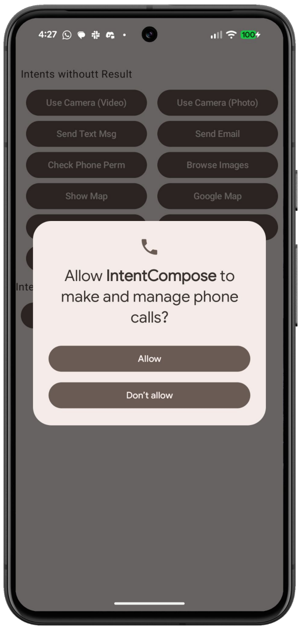
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
   <uses-permission android:name="android.permission.CALL_PHONE" />
   <uses-permission android:name="android.permission.XXXX_YYYY" />
   <application android:allowBackup="true">
       <activity
           android:name=".MainActivity"
           android:exported="true"
           android:label="@string/app_name">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
   </application>

</manifest>
```

Permissions & User Consent

```
// In app build.gradle.kts
// Permission handling
implementation("com.google.accompanist:accompanist-permissions:0.37.3")
```

```
@Composable
fun YourScreenHere() {
  val phonePermState = rememberPermissionState(Manifest.permission.CALL_PHONE)
  Button(onClick = {
     if (phonePermState.isGranted) {
        val callIntent = Intent(Intent.ACTION_CALL).apply {
           data = Uri.parse(“tel:616-331-9999”)
        }
        context.startAcvity(callIntent)
     } else {
        phonePermState.launchPermissionRequest()
     }
  }) { Text(“Call 331-9999”) }
}
```



Intents with Results

Launching Target Activity

```
// Intent without result
Button(onClick = {
    val myIntent = Intent(Intent.ACTION_XXX).apply {
       // prepare extra data if necessary
    }
    context.startActivity(myIntent)
  }) { Text("Go") }
}
```

```
// Intent with result
val myContract = ActivityResultContracts.StartActivityForResult()
val myLauncher = rememberLauncherForActivityResult(myContract){ result ->
     // Process result here
}
Button(onClick = {
    val myIntent = Intent(Intent.ACTION_XXX).apply {
        // prepare extra data if necessary
    }
    myLauncher.launch(myIntent)
  }) { Text("Go") }
}
```

Image Capture Using Camera Activity

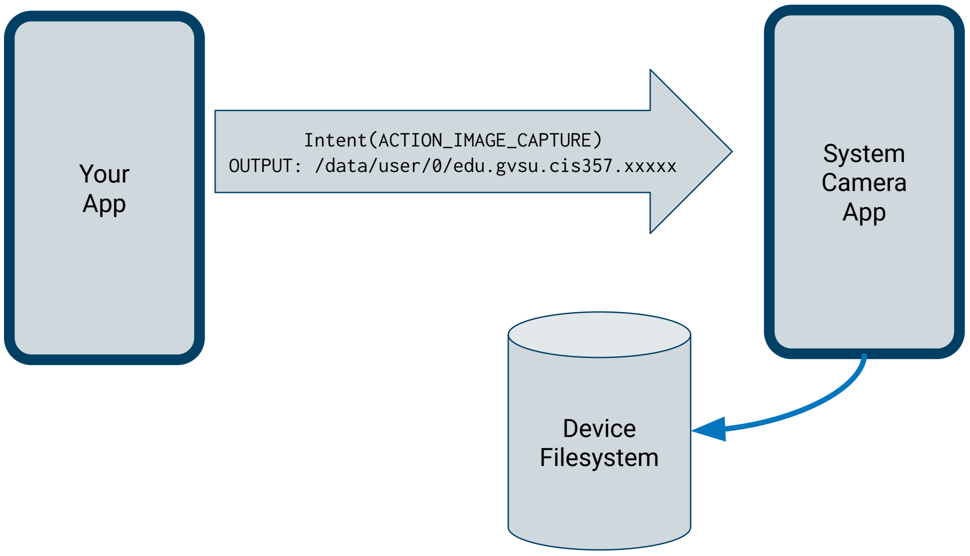
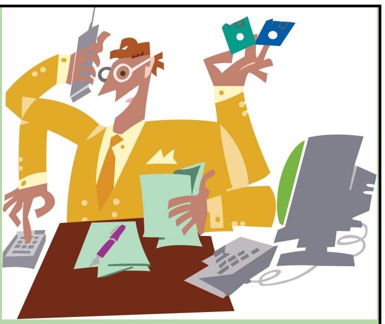

Your App
Intent(ACTION_IMAGE_CAPTURE)
OUTPUT: /data/user/0/edu.gvsu.cis357.xxxxx
System Camera App
Device Filesystem

Image Capture Using Camera Activity

```
// Intent with result
val myContract = ActivityResultContracts.StartActivityForResult()
val myLauncher = rememberLauncherForActivityResult(myContract){ result ->
  if (result.resultCode == RESULT_OK) {
    // The user captured a new image
  } else {
    // The user cancels image capture
  }
}
Button(onClick = {
    val captureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE).apply {
        putExtra(MediaStore.EXTRA_OUTPUT, yourImageUri)
    }
    myLauncher.launch(captureIntent)
  }) { Text("Go") }
}
```

Preparing the (destination) file requires more work than capturing the image itself



URI: Uniform Resource Identifier

- A naming convention used for identifying abstract/physical resources

Uniform Resource Identifier	Associated Resource
`tel:616-331-9999`	Phone number
`geo:27.123,-89.443`	Geographical location (longitude, latitude)
`content://contacts/people/42`	A specific record in contact database
`content://edu.gvsu.cs357.demo/images/123.jpg`	A JPG image

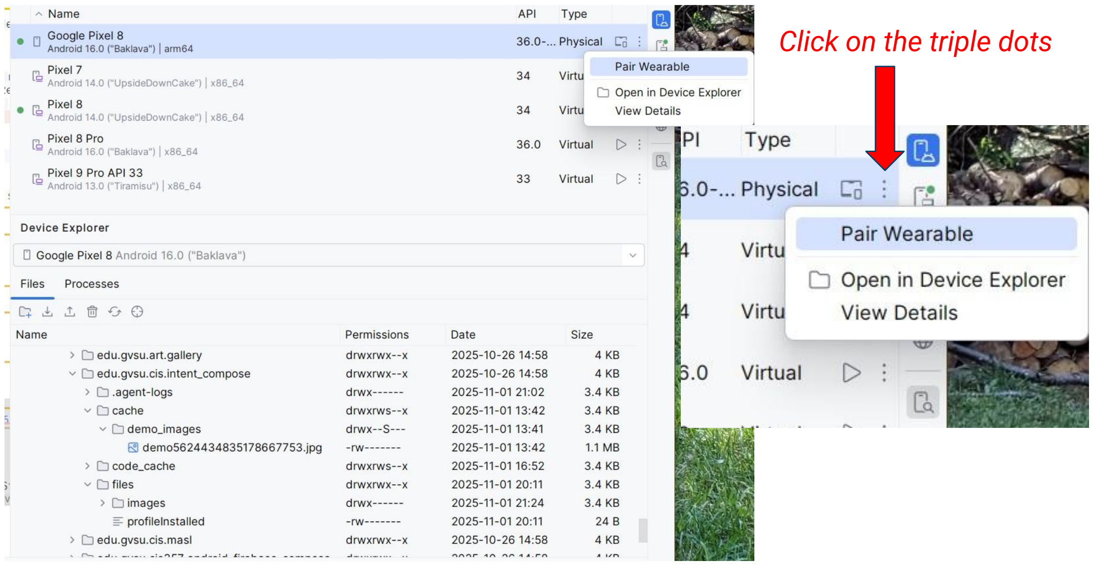
- The file URI decouples the resource identify from the **actual location of the file** in the file system
    - Files can be shared to other apps (via their URI) without exposing the physical location

Android File System

- Android File system = Linux FS
- Five predefined subdirectories

Path Getter Function	Type	App Specific	Removed When App is Uninstalled
`getFilesDir()`	Internal	Yes	Yes
`getCacheDir()`	Internal	Yes	Yes
`getExternalFilesDir()`	External	Yes	Yes
`getExternalCacheDir()`	External	Yes	Yes
Media storage	External	No	No

Android Studio: Emulator File Explorer



```
Name                                                          API       Type
Google Pixel 8
Android 16.0 ("Baklava") | arm64                              36.0-...  Physical
Pixel 7
Android 14.0 ("UpsideDownCake") | x86_64                      34        Virtu
Pixel 8
Android 14.0 ("UpsideDownCake") | x86_64                      34        Virtu
Pixel 8 Pro
Android 16.0 ("Baklava") | x86_64                             36.0      Virtual
Pixel 9 Pro API 33
Android 13.0 ("Tiramisu") | x86_64                            33        Virtual
```

Click on the triple dots

```
Device Explorer
Google Pixel 8 Android 16.0 ("Baklava")
Files   Processes

Name                                        Permissions   Date               Size
  > edu.gvsu.art.gallery                    drwxrwx--x    2025-10-26 14:58   4 KB
  v edu.gvsu.cis.intent_compose             drwxrwx--x    2025-10-26 14:58   4 KB
    > .agent-logs                           drwx------    2025-11-01 21:02   3.4 KB
    v cache                                 drwxrws--x    2025-11-01 13:42   3.4 KB
      v demo_images                         drwx--S---    2025-11-01 13:41   3.4 KB
          demo5624434835178667753.jpg       -rw-------    2025-11-01 13:42   1.1 MB
    > code_cache                            drwxrws--x    2025-11-01 16:52   3.4 KB
    v files                                 drwxrws--x    2025-11-01 20:11   3.4 KB
      > images                              drwx------    2025-11-01 21:24   3.4 KB
        profileInstalled                    -rw-------    2025-11-01 20:11   24 B
  > edu.gvsu.cis.masl                       drwxrwx--x    2025-10-26 14:58   4 KB
```

File Provider

- A special type of Content Provider, but specifically for files
- Purpose: mapping symbolic URI names to physical path on the file system
- Defined using <provider> XML block in AndroidManifest.xml

<provider>

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
   <application android:allowBackup="true">
       <sctivity android:name="_______"></activity>
       <provider
           android:authorities="${applicationId}.provider"
           android:name="androidx.core.content.FileProvider"
           android:exported="false"
           android:grantUriPermissions="true">
           <meta-data android:name="android.support.FILE_PROVIDER_PATHS"
               android:resource="@xml/image_file_paths" />
       </provider>
   </application>
</manifest>
```

```
<!-- res/xml/image_file_path.xml -->
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
   <cache-path  name="my_demo"  path="demo_images"/>
   <files-path  name="my_images"  path="images"/>
</paths>
```

Using FileProvider Programmatically

```
fun setupImageUri() {
    val ctx = app.applicationContext
    val imagePath = File(ctx.filesDir, "images")
    if (!imagePath.exists())
        imagePath.mkdirs()

    val tmpFile = File.createTempFile("cs357", ".jpg", imagePath)
    val imageFileUri = FileProvider.getUriForFile(
           ctx, ctx.packageName + ".provider", tmpFile)
}
```

- File name: `cs3570923807656234.jpg`
- Linux path: `/data/user/0/edu.gvsu.cis356.demo/files/images/cs3570923807656234.jpg`
- URI: `content://edu.gvsu.cis356.demo.provider/my_images/cs3570923807656234.jpg`

Receiving System Broadcast

Broadcast Messages

- ACTION_ACCOUNT_REMOVED
- ACTION_AIRPLANE_MODE_CHANGED
- **ACTION_BATTERY_CHANGE**, ACTION_BATTERY_LOW
- ACTION_DATE_CHANGED
- ACTION_HEADSET_PLUG
- ACTION_SCREEN_OFF, ACTION_SCREEN_ON
- ...and *many more*

Broadcast Receiver in ViewModel

```
class AppViewModel(val app: Application): AndroidViewModel(app) {
   private val XYZReceiver = object: BroadcastReceiver() {
       override fun onReceive(ctx: Context?, intent: Intent?) {
           // Do the work here
       }
   }

   init {
       app.applicationContext.registerReceiver(XYZReceiver,
               IntentFilter(Intent. ACTION_XYZ))
   }

   override fun onCleared() {
       super.onCleared()
       app.applicationContext.unregisterReceiver(XYZReceiver)
   }
}
```

Broadcast Receiver Example: Battery Level

```
class AppViewModel(val app: Application): AndroidViewModel(app) {
   private val _batteryLevel = MutableStateFlow(0)
   val batteryLevel = _batteryLevel.asStateFlow().debounce(timeoutMillis = 1000)

   private val batteryChangeReceiver = object: BroadcastReceiver() {
       override fun onReceive(ctx: Context?, intent: Intent?) {
           val level = intent?.getIntExtra(BatteryManager.EXTRA_LEVEL, 0)
           _batteryLevel.update { level ?: -1 }
       }
   }

   init {
       app.applicationContext.registerReceiver(batteryChangeReceiver,
               IntentFilter(Intent.ACTION_BATTERY_CHANGED))
   }

   override fun onCleared() {
       super.onCleared()
       app.applicationContext.unregisterReceiver(batteryChangeReceiver)
   }
}
```