ANDROID INTENTS AND INTENT FILTERS

INTENT I

Intents are used for switching between Activities (Similar to iOS Segues). It can be used for numerous other things as well, like starting a service or find other programs that can handle a certain document.

Android distinguishes between:

- Explicit Intents: The developer defines what class should be executed ("start class DetailActivity", already covered).
- Implicit Intents: The system defines what class to be executed based on the information provided by the developer (developer provides PDF Document, system starts favourite PDF Reader).

IMPLICIT INTENTS I

- An explicit Intent defines the target of the operation (called component) explicitly (typically, by providing a class to call).
- An implicit Intent has not defined a component. Instead, it must include enough information for the system to determine which of the available components is best to run for that Intent. This is typically done by providing an action (ie. ACTION_VIEW) and data (ie. a URL).

IMPLICIT INTENTS II

```
//add onClick listener for our button
urlButton.setOnClickListener(View.OnClickListener {
    // Implicit Intent by specifying a URL
    val i = Intent(
        Intent.ACTION_VIEW,
        Uri.parse("http://www.tagesanzeiger.ch")
    )

    // Starts Implicit
    startActivity(i)
})
```

INTENTS WITH RESULTS I

```
//Some intents return a result. One example is the camera app
//that returns the taken picture. Use it by replacing
//startActivity with startActivityForResult. Afterwards,
//the method onActivityResult will be called automatically

//use an id to identify the call
val REQUEST_IMAGE_CAPTURE = 1
...
startCamera.setOnClickListener(View.OnClickListener {
 val pictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)

 if (pictureIntent.resolveActivity(requireContext().packageManager)
     != null) {
   startActivityForResult(pictureIntent, REQUEST_IMAGE_CAPTURE)
 }
})
```

INTENTS WITH RESULTS II

```
override fun onActivityResult(requestCode: Int,
   resultCode: Int, data: Intent?) {
   //this method might be called by other activities with
   //result. Therefore, check the provided id
 if (requestCode === REQUEST_IMAGE_CAPTURE) {
   //a thumbnail of the picture is stored in the intent
   //parameter named "data"
   val bitmap = data?.extras!!.get("data") as Bitmap
   //do something with the bitmap (ie present it in
   //an ImageView)
   imageView.setImageBitmap(bitmap)
 }
}
```

PENDINGINTENT

- A PendingIntent wraps another intent object.
- It is passed to a foreign application. This application gets the right to perform the operation (the wrapped intent).
- For security reasons, a PendingIntent should always wrap an explicit intent (define the class to call).

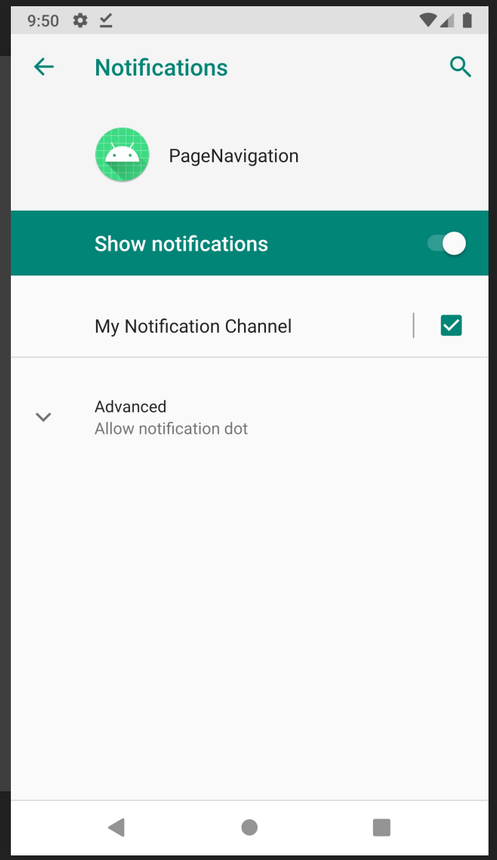
PENDING INTENT FOR NOTIFICATIONS

- Can be used to inform a user outside of your application.
- Is presented in the notification area.
- Can have multiple actions, you can even style dialogs.
- Since Android 8, you need to use one or more channels for your notifications.

CREATE NOTIFICATION CHANNEL

```
fun createNotificationChannel() : NotificationChannel {
  val name = "My Notification Channel" //channel name
  val descriptionText = "This is the description text"
  val importance = NotificationManager.IMPORTANCE_DEFAULT
  val channel = NotificationChannel("MY_CHANNEL", name,
    importance)
  channel.description = descriptionText
  //Register the channel with the system. Not also the
  //kotlin way to provide a class: "::class.java"
  val notificationManager = getSystemService(
   requireContext(), NotificationManager::class.java)
  notificationManager?.createNotificationChannel(channel)

  return channel
}
```



CREATE PENDING INTENT

```
val channel = createNotificationChannel()
//activity to open when the user clicks on the notification
val i = Intent(requireContext(), MainActivity::class.java)
val pendingIntent = PendingIntent.getActivity(
    requireContext(), 0,
    i, 0
)
//build it, add a text and an icon
val builder = Notification.Builder(requireContext(), channel.id)
    .setSmallIcon(android.R.drawable.btn_plus)
    .setContentTitle("Our notification")
    .setContentIntent(pendingIntent)
//show the notification by calling NotificationManager.notify
val notificationManager = getSystemService(requireContext(),
 NotificationManager::class.java)
notificationManager?.notify("test", 0, builder.build())
```

PERMISSIONS

PERMISSIONS IN ANDROID I

- Every app runs in a sandbox, special capabilities need permissions.
- All permissions are identified by a unique label.
- You can grant/deny permissions for an app in the settings (Settings->Apps->(Choose app)).

PERMISSIONS IN ANDROID II

Using capabilities without permissions will result in an exception:

```
Caused by: java.lang.SecurityException: Permission Denial: opening
provider com.android.providers.contacts.ContactsProvider2 from
ProcessRecord{2c165c26436:ch.zhaw.contactapplication/u0a112}
(pid=6436, uid=10112) requires
android.permission.READ_CONTACTS or
android.permission.WRITE_CONTACTS
```

DECLARING PERMISSIONS

- Until Android 5.1, permissions were only defined in the manifest file. That changed to a dynamic approach where users have to allow a permission during runtime.
- However, you still need to declare your permissions in Android manifest with a element! If you forget the manifest declaration, your permission is automatically declined during runtime.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="ch.zhaw.contactapplication">
 <uses-permission android:name="android.permission.READ_CONTACTS" />
 <application ...
```

DANGEROUS PERMISSIONS I

- Android manages permissions in groups. If you already permit access to READ_CONTACTS, Android will automatically grand WRITE_CONTACTS as both are in the same group.
- Until Android 6, permissions where only declared in the manifest. Since then, Android distinguishes between normal and dangerous protection levels. Normal permissions (ie INTERNET) will be granted automatically when the permission is declared in the manifest, dangerous permissions need to be allowed by the user explicitly.

DANGEROUS PERMISSIONS II

```
val REQUEST_CONTACTS = 1 //define an id for this permission
...
//check if the app already has the permission (to read contacts)
if (ContextCompat.checkSelfPermission(this,
  Manifest.permission.READ_CONTACTS) ===
    PackageManager.PERMISSION_DENIED)
    //no permission -> ask
    ActivityCompat.requestPermissions(this, arrayOf(
    Manifest.permission.READ_CONTACTS), REQUEST_CONTACTS)
else
   readContacts() //read the contacts if you have the permission
```

DANGEROUS PERMISSIONS III

The result of requestPermissions can be evaluated:

```
//this method is called when the user has chosen to allow or
//deny a permission (after requestPermissions)
override fun onRequestPermissionsResult(requestCode: Int,
  permissions: Array<out String>, grantResults: IntArray) {
   super.onRequestPermissionsResult(requestCode,
     permissions, grantResults)
   if (requestCode == REQUEST_CONTACTS) {
    if (grantResults.size > 0
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
     readContacts();
    }
   }
}
```

READ CONTACTS I

```
//the contacts are read with a query and a cursor (similar
//to sql database reading)
protected fun readContacts() {
  val uri: Uri = ContactsContract.Contacts.CONTENT_URI
  //the array defines the fields to be returned
  val cursor: Cursor? = contentResolver.query(uri, arrayOf(
    ContactsContract.Contacts._ID,
    ContactsContract.Contacts.DISPLAY_NAME,
    ContactsContract.Contacts.HAS_PHONE_NUMBER),
    null, null, null)
```

READ CONTACTS II

```
  if (cursor != null) {
    var res: Boolean = cursor.moveToFirst()
    while (res) {
      //read the id (id == 0) first
      val id  = cursor.getString(0)
      //the name (id == 1)
      var text: String = cursor.getString(1)
      //unfortunately, there is no "getBool"
      if (cursor.getString(2) == "1") {
        text += " " + getPhoneNumberForUser(id)
      }
      Log.i("ContactExample", text)
      res = cursor.moveToNext()
    }
    cursor.close()
}}
```

READ CONTACTS III

```
//the phone number needs to be retrieved with a different call
protected fun getPhoneNumberForUser(contactId: String): String? {
 var number = ""
 val cursor = contentResolver.query(
  ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
  ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " +
  contactId, null, null)
 if (cursor!!.moveToFirst()) {
    while (!cursor.isAfterLast) {
        number += cursor.getString(cursor.getColumnIndex(
          ContactsContract.CommonDataKinds.Phone.NUMBER)) + " "
        cursor.moveToNext()
    }
 }
 cursor.close()
 return number}
```

FILE SHARING

Accessing the same file from different apps is prohibited. If one app wants to allow other apps to access their file(s), it needs to define a FileProvider.

FILEPROVIDER

The FileProvider class enables secure sharing of files associated with an app by creating a content:// Uri for a file instead of a file:/// Uri. A content URI can be grant read and write access using temporary access permissions (Use Intent.setFlags() to add temporary permissions).

FILEPROVIDER CONFIGURATION I

Add an entry in the manifest file:

```
<application ...>
...
<provider
  <!-- name of the FileProvider class -->
  android:name=".GenericFileProvider"
  android:authorities="${applicationId}.provider"
  android:exported="false"
  android:grantUriPermissions="true">
  <meta-data
    <!-- name and place of the xml document -->
    android:name="android.support.FILE_PROVIDER_PATHS"
    android:resource="@xml/provider_paths"/>
</provider>
</application>
```

FILEPROVIDER CONFIGURATION II

XML Document:

```
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <cache-path name="cache_files" path="."/>
</paths>
```

FileProvider Class:

```
class GenericFileProvider : FileProvider() {}
```

FILEPROVIDER MAPPING

XML	Value from	Desc
<files-path/>	Context.getFilesDir()	files directory of internal storage
<cache-path/>	Context.getCacheDir()	cache directory of internal storage
<external-path/>	Environment.getExternalStorageDirectory()	Deprecated
<external-files-path/>	Context.getExternalFilesDir(String)	files (not root) directory of external storage (SD card)
<external-cache-path/>	Context.getExternalCacheDir()	cache directory of external storage
<external-media-path/>	Context.getExternalMediaDirs()	media directory of external storage

FILEPROVIDER AND CAMERA INTENT

```
//create the intent
val pictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
//define where the image should be stored from the camera
//val dataDir = requireContext().filesDir
val dataDir = requireContext().cacheDir

//create the image file in this directory
val f= File (dataDir, "image.jpg")
//retrieve the uri from the file provider (note that a matching
//path entry needs to be defined in the FileProvider xml document)
val fUri = FileProvider.getUriForFile(requireContext(),
requireContext().applicationContext.packageName + ".provider", f);
//don't forget to add this flag as the intent needs write
//permission to your file
pictureIntent.addFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION)
pictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, fUri);
```

CAMERA INTENT RESULT

```
override fun onActivityResult(requestCode: Int, resultCode: Int,
  data: Intent?) {
  if (requestCode === REQUEST_IMAGE_CAPTURE) {
   //create the file name again (or use a member variable)
   val dataDir = requireContext().cacheDir
   val f = File (dataDir, "image.jpg")

   //read the file and create a Bitmap
   val bitmap = BitmapFactory.decodeFile(f.absolutePath)
   //do whatever with the bitmap, ie present it in an ImageView
   imageView.setImageBitmap(bitmap)
  }
}
```

DEVICE COMPATIBILITY

- Android automatically filters your app based on your permissions (at least for some of them, see list here). If, for example, you declare a camera permission, your app is filtered from devices that do not have a camera.
- It is strongly recommended that you declare hardware features using uses-feature and [uses-sdk] in your manifest.
- You can finegrain hardware features. For example, you can declare an autofocus camera. You can also define that a hardware feature is not mandatory using the android:required attribute.

FURTHER COMPONENTS

SCROLLVIEW I

- The ScrollView offers scrolling functionality to the child component (only one child allowed).
- A lot of Apps implement a ScrollView in any page layout. This prevents from the problem of not using an app if there is too much content for the page. Keep in mind that users can change the default text size.
- Only vertical scrolling is supported.

SCROLLVIEW II

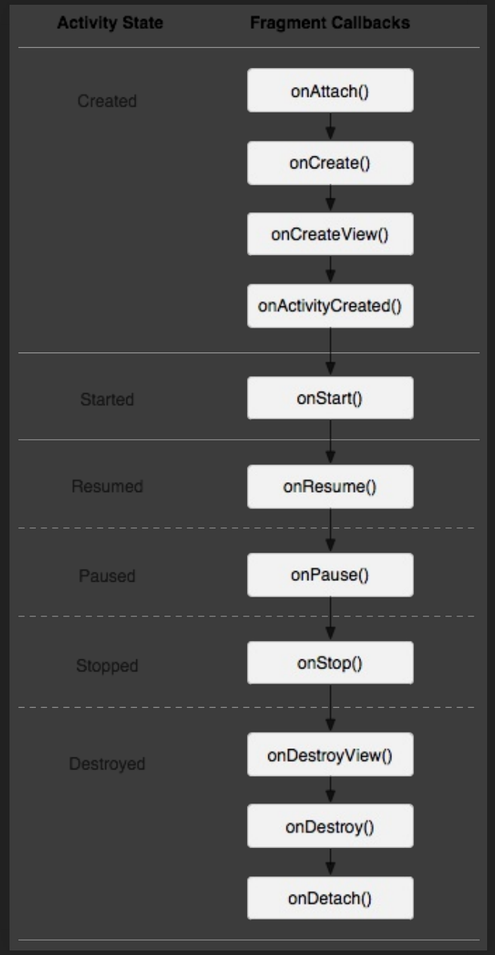
```
<!-- set width and height to 0dp (Use ConstraintLayout) -->
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="0dp"
  android:layout_height="0dp"
  android:fillViewport="true"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintLeft_toLeftOf="parent"
  app:layout_constraintRight_toRightOf="parent"
  app:layout_constraintTop_toTopOf="parent">
    ...
</ScrollView>
```

FRAGMENT

Fragments are:

- building blocks (small UI components)
- used to encapsulate UI components of an activiy
- managed by the owning activity
- easily reused
- typically reordered to support different devices (phone / tablet).
  http://developer.android.com/guide/components/fragments.html

FRAGMENT LIFECYCLE I



Source: developer.android.com

FRAGMENT LIFECYCLE II

Android recommends to implement at least the following three methods in each fragment:

- onCreate: Called when the Fragment is created (only once within a lifecycle). Create all components that should be retained during Pause/Resume.
- onCreateView: Called when the Fragment is drawn. Return the Fragment layout.
- onPause: Called when the Fragment is left by the user. In contrast to an Activity, use this method to persist your data (see here).

Use onViewCreated to initialize the components.

FRAGMENT CREATION I

```
class FirstFragment : Fragment() { //inherit from Fragment

 override fun onCreateView(
   inflater: LayoutInflater, container: ViewGroup?,
   savedInstanceState: Bundle?
 ): View? {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_first,container,false)
 }

 override fun onViewCreated(view: View,savedInstanceState: Bundle?){
      super.onViewCreated(view, savedInstanceState)
       button.setOnClickListener {...}
 }
}
```

FRAGMENT CREATION II

```
<!-- Add a layout file (fragment_first.xml) -->
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout  ...>

    <TextView
        android:id="@+id/textview_first"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_first_fragment"
        app:layout_constraintBottom_toTopOf="@id/button_first"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

HOST FRAGMENT I

- NavHostFragment provides an area where different fragments could be presented including a navigation between them.
- As the name suggests, NavHostFragment is itself a fragment. Therefore, it can be included in a layout as follows:

```
<fragment
  android:name="androidx.navigation.fragment.NavHostFragment"
  app:defaultNavHost="true"
  app:navGraph="@navigation/nav_graph"
  ... />
```

HOST FRAGMENT II

- android:name: Provide a class name for a fragment to create, in this case the NavHostFragment.
- app:defaultNavHost: Has to be set for back button support.
- app:navGraph: references the nav_graph xml file that stores all fragments to be displayed within in this host.

HOST FRAGMENT III

```
<!-- host fragment within an activity layout-->
 <fragment
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:defaultNavHost="true"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:navGraph="@navigation/nav_graph" />
```

HOST FRAGMENT IV

```
<!-- create a nav_graph that defines the fragments of this host -->
<navigation ...
    android:id="@+id/nav_graph"
    app:startDestination="@id/FirstFragment"> <!-- default -->
    <fragment
        android:id="@+id/FirstFragment"
        android:name="ch.zhaw.fragmentexample.FirstFragment"
        android:label="@string/first_fragment_label">
        <action
            android:id="@+id/action_FirstFragment_to_SecondFragment"
            app:destination="@id/SecondFragment" />
    </fragment>
    <fragment
        android:id="@+id/SecondFragment" ...
    </fragment>
</navigation>
```

HOST FRAGMENT NAVIGATION

Navigate between Fragments by defining the action in the nav_graph:

```
<action
    android:id="@+id/action_FirstFragment_to_SecondFragment"
    app:destination="@id/SecondFragment" />
```

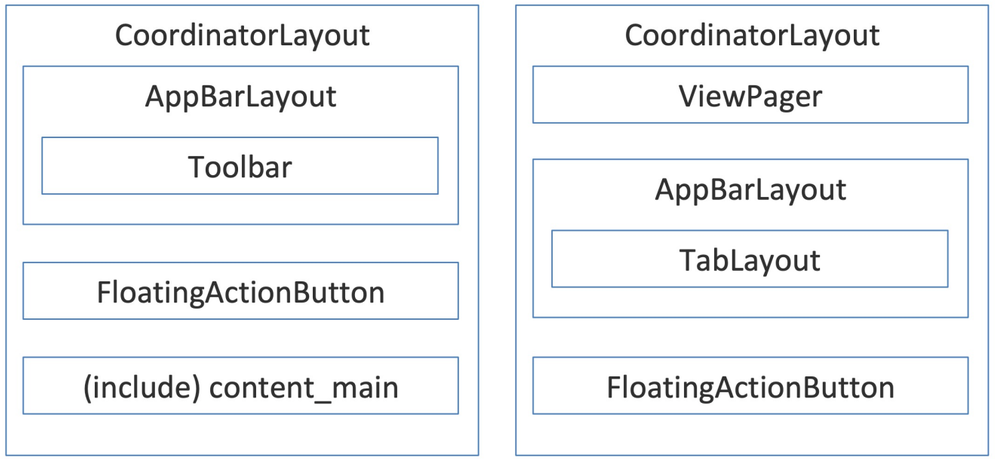
and using the action within the fragment class:

```
findNavController()
        .navigate(R.id.action_FirstFragment_to_SecondFragment)
```

FRAGMENTS AND TAB BAR

- Fragments can be used to show one page of a tab bar.
- A good base of a tab bar is a pager (ViewPager), as it allows the user to swipe through the tabs.
- The tab bar itself can be added to the pager by attaching the TabLayout component on top.

TAB LAYOUT I



TAB LAYOUT II

```
<!-- The default activity layout is as follows: -->
<androidx.coordinatorlayout.widget.CoordinatorLayout ...
    <com.google.android.material.appbar.AppBarLayout ...
        <androidx.appcompat.widget.Toolbar ... />
    </com.google.android.material.appbar.AppBarLayout>
    <...FloatingActionButton ... />
    <include layout="@layout/content_main" />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
<!-- For tab support, add the ViewPager and TabLayout: -->
<androidx.coordinatorlayout.widget.CoordinatorLayout ...
    <androidx.viewpager.widget.ViewPager ... />
    <com.google.android.material.appbar.AppBarLayout ...
        <com.google.android.material.tabs.TabLayout ... />
    </com.google.android.material.appbar.AppBarLayout>
    <...FloatingActionButton ... />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

TAB LAYOUT INIT I

```
//use a FragmentPagerAdapter to support the correct
//tabs/fragments (see next slide).
val sectionsPagerAdapter =
   SectionsPagerAdapter(this, supportFragmentManager)
//init the view pager (using an adapter)
viewPager.adapter = sectionsPagerAdapter
//init the tablayout
tabLayout.setupWithViewPager(viewPager)
```

TAB LAYOUT INIT II

```
class SectionsPagerAdapter(private val context: Context,
 fm: FragmentManager) : FragmentPagerAdapter(fm) {
 override fun getItem(position: Int): Fragment {
  // return the corresponding fragment
  return PlaceholderFragment.newInstance(position + 1)
 }
 override fun getPageTitle(position: Int): CharSequence? {
  private val TAB_TITLES =
    arrayOf(R.string.tab_text_1, R.string.tab_text_2)
  //and title for this tab
  return context.resources.getString(TAB_TITLES[position])
 }
 override fun getCount(): Int {
  //how many tabs are there?
  return 2
}}
```