Mobile Applications (420-P84-AB)

John Abbott College

Misc

—

Aakash Malhotra

Receive System Event Broadcasts

Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern. These broadcasts are sent when an event of interest occurs. For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging.

Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.

Generally speaking, broadcasts can be used as a messaging system across apps and outside of the normal user flow.

Receive System Event Broadcasts

The broadcast message itself is wrapped in an `Intent` object whose action string identifies the event that occurred (for example `android.intent.action.AIRPLANE_MODE`). The intent may also include additional information bundled into its extra field. For example, the airplane mode intent includes a boolean extra that indicates whether or not Airplane Mode is on.

Receive System Event Broadcasts

Apps can receive broadcasts in two ways: through manifest-declared receivers and context-registered receivers.

If your app targets API level 26 or higher, you cannot use the manifest to declare a receiver for implicit broadcasts (broadcasts that do not target your app specifically)

Receive System Event Broadcasts

To register a receiver with a context, perform the following steps:

1. Create an instance of `BroadcastReceiver`.

   ```
   BroadcastReceiver br = new MyBroadcastReceiver();
   ```

2. Create an `IntentFilter` and register the receiver by calling `registerReceiver(BroadcastReceiver, IntentFilter)`:

   ```
   IntentFilter filter = new
   IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);

   filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);

   this.registerReceiver(br, filter);
   ```

Receive System Event Broadcasts

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
    // Do something here
 }
}
```

Android’s guidelines for UI and UX

- Providing Density-Specific Icon Sets
- Use common naming conventions for icon assets
- Set up a working space that organizes files for multiple densities
- Use vector shapes where possible
- Start with large artboards
- Make sure that corresponding assets for different densities use the same filenames
- Be conservative with minWidth and minHeight

Write to Internal Storage

```
String FILE_NAME = "file.txt";
try {
    FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
    fos.write(someText.toString().getBytes());
    fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Read to Internal Storage

```
try {
    BufferedReader bReader = new BufferedReader(new
InputStreamReader(openFileInput(FILE_NAME)));
    String line;
    StringBuffer text = new StringBuffer();
    while ((line = bReader.readLine()) != null) {
        text.append(line + "\n");
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Services

A `Service` is an application component that can perform long-running operations in the background, and it doesn't provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application. For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:

Foreground

> A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services must display a Notification. Foreground services continue running even when the user isn't interacting with the app.

Services

Background

A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

Bound

A service is *bound* when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed

Content Providers

A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database. A row represents an instance of some type of data the provider collects, and each column in the row represents an individual piece of data collected for an instance.

A content provider coordinates access to the data storage layer in your application for a number of different APIs and components, these include:

- Sharing access to your application data with other applications
- Sending data to a widget
- Returning custom search suggestions for your application through the search framework using `SearchRecentSuggestionsProvider`
- Synchronizing application data with your server using an implementation of `AbstractThreadedSyncAdapter`
- Loading data in your UI using a `CursorLoader`

References:

- https://developer.android.com/guide/practices/ui_guidelines/
- https://developer.android.com/guide/practices/ui_guidelines/icon_design
- https://developer.android.com/guide/practices/ui_guidelines/widget_design
- https://developer.android.com/guide/components/broadcasts
- https://en.proft.me/2014/06/21/how-readwrite-files-android/
- https://developer.android.com/guide/components/services
- https://developer.android.com/guide/topics/providers/content-provider-basics