Android Services & Local IPC: Communicate from Started Services to Activities via Broadcast Receivers

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

- Understand how to use Broadcast Receivers to communicate from Started Services back to their invoking Activities
- Supports IPC with (multiple) remote processes without using AIDL
Overview of Broadcast Receivers

- Broadcast Receivers are components (receivers) that register for broadcast events & receive/react to the events

[Diagram showing the interaction between Phone App, System Server, Activity Manager Service, and Broadcast Receivers]

developer.android.com/reference/android/content/BroadcastReceiver.html
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/react to the events
- Events implemented as Intents

![Diagram of Broadcast Receivers]

1: Register for broadcast intent
2: Detect that battery is low & create corresponding intent
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/react to the events
  - Events implemented as Intents
  - Events are broadcast system-wide

3: Call sendBroadcast() to inform interested receivers that battery is low
Overview of Broadcast Receivers

- BroadcastReceivers are components (*receivers*) that register for broadcast events & receive/ react to the events
  - Events implemented as Intents
  - Events are broadcast system-wide
  - When an event occurs the Intents are disseminated to all matching receivers via their `onReceive()` hook methods

4: `onReceive()` called back to report low battery
3: Call `sendBroadcast()` to inform interested receivers that battery is low
Overview of Broadcast Receivers

- Broadcast Receivers are components (receivers) that register for broadcast events & receive/react to the events.
- Activities can create receivers that register for system or app events.

Additional resources:

www.vogella.com/articles/AndroidBroadcastReceiver/article.html has more information.
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/act on the events.
- Activities can create receivers that register for system or app events.
- A receiver is restricted on what it can do when it handles an Intent.
  - e.g., it may not show a dialog or bind to a service.

[Link to developer.android.com reference page](developer.android.com/reference/android/content/BroadcastReceiver.html#ReceiverLifecycle)
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/react to the events
- Activities can create receivers that register for system or app events
- A receiver is restricted on what it can do when it handles an Intent
- Two ways to register a receiver:
  - Statically publish it via the <receiver> tag in the AndroidManifest.xml file

```
<receiver android:name="PhoneApp$NotificationBroadcastReceiver"
  exported="false">
  <intent-filter>
    <action android:name="com.android.phone.ACTION_HANG_UP_ONGOING_CALL" />
    <action android:name="com.android.phone.ACTION_SEND_SMS_FROM_NOTIFICATION"/>
  </intent-filter>
</receiver>
```
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/react to the events.
- Activities can create receivers that register for system or app events.
- A receiver is restricted on what it can do when it handles an Intent.
- Two ways to register a receiver:
  - Statically publish it via the `<receiver>` tag in the AndroidManifest.xml file.
  - Dynamically register it with Context.registerReceiver().

```java
final BroadcastReceiver mReceiver =
    new PhoneAppBroadcastReceiver();
...
IntentFilter intentFilter =
    new IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
...
registerReceiver(mReceiver, intentFilter);
```
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/react to the events.
- Activities can create receivers that register for system or app events.
- A receiver is restricted on what it can do when it handles an Intent.
- Two ways to register a receiver.
- Android supports several broadcast mechanisms:
  - Normal – Sent with Context.sendBroadcast(), which is completely asynchronous.

[Diagram showing Phone App, System Server, and Activity Manager Service connected with arrows to represent broadcast receivers]
Overview of Broadcast Receivers

- BroadcastReceivers are components (receivers) that register for broadcast events & receive/react to the events.
- Activities can create receivers that register for system or app events.
- A receiver is restricted on what it can do when it handles an Intent.
- Two ways to register a receiver.
- Android supports several broadcast mechanisms.
  - *Normal* – Sent with `Context.sendBroadcast()`, which is completely asynchronous.
  - *Ordered* – Sent with `Context.sendOrderedBroadcast()`, which is delivered to one receiver at a time.

[Link to documentation](http://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String))
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action.
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity.

The Intent stores the URI as an “data”. 

![Diagram showing the interaction between DownloadActivity, BroadcastReceiver, and IntentFilter.

1. DownloadActivity initiates a download.
2. It creates an Intent and registers a BroadcastReceiver.
3. BroadcastReceiver receives the ACTION_COMPLETE action.
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETED action.
- DownloadService broadcasts an ACTION_COMPLETED back to the Activity.
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action.
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity.
Using Broadcast Receiver in Download App

- DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
- DownloadService broadcasts an ACTION_COMPLETE back to the Activity

DownloadActivity

- initiateDownload() 4
- onCreate() 3
- onResume() 2
- registerReceiver() 1

IntentFilter

BroadcastReceiver

onReceive() 1

Intent

ActivityManagerService

onCreate() 6

onStartCommand() 5

sendMessage() 7

downloadImage() 8

DownloadService

StartService() 3
Using Broadcast Receiver in Download App

• DownloadActivity creates & registers a BroadcastReceiver with an IntentFilter configured with the ACTION_COMPLETE action
• DownloadService broadcasts an ACTION_COMPLETE back to the Activity
Programming a Broadcast Receiver in Activity

• DownloadActivity contains a BroadcastReceiver instance with hook method

```java
public class DownloadActivity extends Activity {
    private BroadcastReceiver onEvent = null;
    public void onCreate(Bundle onSavedInstanceState) {
        onEvent = new BroadcastReceiver() {
            public void onReceive(Context context, Intent intent) {
                String path = intent.getStringExtra(RESULT_PATH);
                Receive Intent sent by sendBroadcast()
                Extract the path using “extra” within the Intent
                if (path == null)
                    Toast.makeText(DownloadActivity.this,
                        "Download failed.", Toast.LENGTH_LONG).show();
                    displayImage(path);
                }
            }
        };
        ...  
    }
```
Programming a Broadcast Receiver in Activity

- DownloadActivity's lifecycle methods register & unregister the receiver

```java
public class DownloadActivity extends Activity {
    ...
    public void onResume() {
        super.onResume();
        IntentFilter filter =
            new IntentFilter(ACTION_COMPLETE);
        registerReceiver(onEvent, filter);
    }

    public void onPause() {
        super.onPause();
        unregisterReceiver(onEvent);
    }
    ...
```

Register BroadcastReceiver when Activity resumes

Unregister BroadcastReceiver before Activity pauses
Programming a Broadcast Receiver in Activity

• DownloadActivity passes the package name to the DownloadService

```java
class DownloadActivity extends Activity {
    ...

    public void initiateDownload(View v) {
        Intent intent = new Intent(DownloadActivity.this, DownloadService.class);
        ...
        intent.putExtra(PACKAGE_NAME, getPackageName());
        startService(intent);
    }
    ...
    Start the service
}
```
Programming a Broadcast Receiver in Service

- DownloadService replies to DownloadActivity via sendBroadcast()

```java
public class DownloadService extends Service {
    ...  
    private final class ServiceHandler extends Handler {
        ...  
        public void downloadImage(Intent intent) {
            // ...  
            Code to downloading image to pathname goes here

            Intent replyIntent = new Intent(ACTION_COMPLETE);
            replyIntent.putExtra(RESULT_PATH, pathname);
            String packageName = intent.getStringExtra(PACKAGE_NAME);
            intent.setPackage(packageName);
            sendBroadcast(replyIntent);
        }
        ...  
        Broadcast pathname to Activity
    }
    ...
```
Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android.
- Broadcast Receivers are generally used for more interesting use-cases...

[Diagram showing the processes and methods involved in using Broadcast Receivers and Services.]

developer.android.com/reference/android/content/BroadcastReceiver.html#Security
Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android
- However, there are subtle issues with security

[developer.android.com/reference/android/content/BroadcastReceiver.html#Security]
Summary

• Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android

• However, there are subtle issues with security
  • The Intent namespace is global
    • This may cause subtle conflicts
Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android.
- However, there are subtle issues with security.
  - The Intent namespace is global.
  - `registerReceiver()` allows any app to send broadcasts to that registered receiver.
  - Use permissions to address this.

[developer.android.com/reference/android/content/BroadcastReceiver.html#Security]
Summary

- Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android.

- However, there are subtle issues with security:
  - The Intent namespace is global.
  - `registerReceiver(BroadcastReceiver, IntentFilter)` allows any app to send broadcasts to that registered receiver.
  - When a receiver is published in an app’s manifest & specifies intent-filters for it, any other app can send broadcasts to it regardless of the specified filters.
  - To prevent others from sending to it, make it unavailable to them with `android:exported="false"

```xml
<receiver
    android:enabled="true | false"
    android:exported="true | false"
    android:icon="drawable resource"
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:process="string">
  ...
</receiver>
```

[developer.android.com/guide/topics/manifest/receiver-element.html](developer.android.com/guide/topics/manifest/receiver-element.html)
Summary

• Broadcast Receivers provide a scalable framework for communicating between (potentially multiple) processes in Android

• However, there are subtle issues with security
  • The Intent namespace is global
  • registerReceiver(BroadcastReceiver, IntentFilter) allows any app to send broadcasts to that registered receiver
  • When a receiver is published in an app’s manifest & specifies intent-filters for it, any other app can send broadcasts to it regardless of the filters that are specified
  • sendBroadcast() et al allow any other app to receive broadcasts
  • Broadcasts can be restricted to a single app with Intent.setPackage()

developer.android.com/reference/android/content/Intent.html#setPackage(java.lang.String)
Android Services & Local IPC: Communicating via Pending Intents

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

- Understand how to use Pending Intents to communicate from (Started) Services back to other components (e.g., Activities, Broadcast Receivers, etc.)
- A PendingIntent is a token given to an App to perform an action on your Apps' behalf *irrespective* of whether your App's process is alive.
Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code
  - e.g., Notification Manager, Alarm Manager, or other 3rd party apps

[Developer documentation link](developer.android.com/reference/android/app/PendingIntent.html)
Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code.
  - e.g., Notification Manager, Alarm Manager, or other 3rd party apps.
- The token maintained by the system represents an Intent & the action to perform on that Intent later.
  - Can be configured to work irrespective of whether the original App process is alive or not.

Start an Activity to read email.

developer.android.com/reference/android/app/PendingIntent.html
Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code.

- Pending Intents can be created via various methods, e.g.:
  - `getActivity()` on PendingIntent
    - The PendingIntent returned by this method starts a new Activity when `send()` is called on it.

```java
public static PendingIntent getActivity(Context context, int requestCode, Intent intent, int flags)
```

Retrieves a PendingIntent that will start a new activity, like calling `Context.startActivity(Intent)`. Note that the activity will be started outside of the context of an existing activity, so you must use the `Intent.FLAG_ACTIVITY_NEW_TASK` launch flag in the Intent.

For security reasons, the Intent you supply here should almost always be an explicit intent, that is specify an explicit component to be delivered to through `setClass(android.content.Context, Class) Intent.setClass`

Parameters:
- `context`: The Context in which this PendingIntent should start the activity.
- `requestCode`: Private request code for the sender (currently not used).
- `intent`: Intent of the activity to be launched.
- `flags`: May be `FLAG_ONE_SHOT, FLAG_NO_CREATE, FLAG_CANCEL_CURRENT, FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

Returns:
Returns an existing or new PendingIntent matching the given parameters. May return null only if `FLAG_NO_CREATE` has been supplied.

[developer.android.com/reference/android/app/PendingIntent.html](http://developer.android.com/reference/android/app/PendingIntent.html)
Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code.
- Pending Intents can be created via various methods, e.g.:
  - `getActivity()` on PendingIntent
  - `getBroadcast()` on PendingIntent
- The Pending Intent returned by this method sends a broadcast to a Receiver when `send()` is called on it.

```java
public static PendingIntent getBroadcast(Context context, int requestCode, Intent intent, int flags)
```

Retrieves a PendingIntent that will perform a broadcast, like calling `Context.sendBroadcast()`. For security reasons, the `Intent` you supply here should almost always be an explicit `Intent`, that is specify an explicit component to be delivered to through `setClass(android.content.Context, Class) Intent.setClass`.

**Parameters**
- `context` The Context in which this PendingIntent should perform the broadcast.
- `requestCode` Private request code for the sender (currently not used).
- `intent` The Intent to be broadcast.
- `flags` May be `FLAG_ONE_SHOT, FLAG_NO_CREATE, FLAG_CANCEL_CURRENT, FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

**Returns**
Returns an existing or new PendingIntent matching the given parameters. May return null only if `FLAG_NO_CREATE` has been supplied.

[developer.android.com/reference/android/app/PendingIntent.html](http://developer.android.com/reference/android/app/PendingIntent.html)
Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code.
- PendingIntents can be created via various methods, e.g.:
  - `getActivity()` on PendingIntent
  - `getBroadcast()` on PendingIntent
  - `getService()` on PendingIntent
- The PendingIntent returned by this method starts a new Service when `send()` is called on it.

```java
public static PendingIntent getService(Context context, int requestCode, Intent intent, int flags)
```

Retrieve a PendingIntent that will start a service, like calling `Context.startService()`. The start arguments given to the service will come from the extras of the Intent.

For security reasons, the `Intent` you supply here should almost always be an `explicit intent`, that is specify an explicit component to be delivered to through `setClass(android.content.Context, Class) Intent.setClass`

Parameters

- **context**: The Context in which this PendingIntent should start the service.
- **requestCode**: Private request code for the sender (currently not used).
- **intent**: An Intent describing the service to be started.
- **flags**: May be `FLAG_ONE_SHOT, FLAG_NO_CREATE, FLAG_CANCEL_CURRENT, FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

Returns

Returns an existing or new PendingIntent matching the given parameters. May return null only if `FLAG_NO_CREATE` has been supplied.

Overview of Pending Intents

- A PendingIntent is a token given by an App to another component that allows it to use the permissions of the App to execute a piece of code.
- PendingIntents can be created via various methods, e.g.:
  - `getActivity()` on PendingIntent
  - `getBroadcast()` on PendingIntent
  - `getService()` on PendingIntent
  - `createPendingResult()` on Activity
- The PendingIntent returned by this method sends data back to the Activity via its method `onActivityResult()`.

```java
public PendingIntent createPendingResult(int requestCode, Intent data, int flags)
```

Create a new PendingIntent object which you can hand to others for them to use to send result data back to your `onActivityResult(int, int, Intent)` callback. The created object will be either one-shot (becoming invalid after a result is sent back) or multiple (allowing any number of results to be sent through it).

**Parameters**

- **requestCode**: Private request code for the sender that will be associated with the result data when it is returned. The sender can not modify this value, allowing you to identify incoming results.
- **data**: Default data to supply in the result, which may be modified by the sender.
- **flags**: May be `PendingIntent.FLAG_ONE_SHOT`, `PendingIntent.FLAG_NO_CREATE`, `PendingIntent.FLAG_CANCEL_CURRENT`, `PendingIntent.FLAG_UPDATE_CURRENT`, or any of the flags as supported by `Intent.fillIn()` to control which unspecified parts of the intent that can be supplied when the actual send happens.

**Returns**

Returns an existing or new PendingIntent matching the given parameters. May return null only if `PendingIntent.FLAG_NO_CREATE` has been supplied.

[developer.android.com/reference/android/app/Activity.html #createPendingResult(int, android.content.Intent, int)]
Using PendingIntent w/AlarmManager Service

- PendingIntents are often used with alarms
- Activity creates & schedules a PendingIntent with the Alarm Service

```java
PendingIntent pi;
AlarmManager mgr;

void onCreate(Bundle b) {
    AlarmManager mgr =(AlarmManager)
        getSystemService(ALARM_SERVICE);
    Intent replyIntent = new Intent();
    ... // Set “extras” in replyIntent
    pi = createPendingResult
        (ALARM_ID, replyIntent, 0);
    mgr.setRepeating
        (AlarmManager.ELAPSED_REALTIME_WAKEUP,
         SystemClock.elapsedRealtime() + PERIOD, PERIOD, pi);
    finish();
}
```

Cause the alarm to restart the Activity when it expires
PendingIntents are often used with alarms

Activity creates & schedules a PendingIntent with the Alarm Service

```java
void setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation) {
    Alarm alarm = new Alarm();
    ...
    alarm.when = triggerAtTime;
    alarm.repeatInterval = interval;
    alarm.operation = operation;
    Message msg = Message.obtain();
    msg.what = ALARM_EVENT;
    ...
    mHandler.sendMessageAtTime(msg, alarm.when);
}
```

**AlarmManager maintains its schedule outside of an App's process, so it can give the App control, even if it has to start up a new process along the way**

[developer.android.com/reference/android/app/AlarmManager.html](http://developer.android.com/reference/android/app/AlarmManager.html)
Using PendingIntent w/AlarmManager Service

- PendingIntents are often used with alarms
  - Activity creates & schedules a PendingIntent with the Alarm Service
  - When the timer expires the Alarm Service sends a reply back to the Activity

```java
class AlarmHandler extends Handler {
    void handleMessage(Message m) {
        ...
        alarm.operation.send();
        ...
    }
}
```
PendingIntents are often used with alarms

- Activity creates & schedules a PendingIntent with the Alarm Service
- When the timer expires the Alarm Service sends a reply back to the Activity
- The Activity is restarted & its onActivityResult() method handles the reply

```java
void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == ALARM_ID) {
        // Do something with data in the Intent
    }
}
```
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.

**Diagram:**

1. **DeferredDownload Activity**
   - create PendingIntent
2. **Pending Intent**
   - deferredDownload()
   - onCreate()
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.

Diagram:

1. DeferredDownloadActivity
2. Pending Intent
3. AlarmManager Service

Schedule an alarm to run at some future point in time.
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.

Diagram:

1. DeferredDownloadActivity
2. PendingIntent
3. AlarmManagerService
4. onCreate()
5. onHandleIntent()
6. notify()
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.

Diagram:
- DeferredDownloadActivity
  - onCreate()
  - deferredDownload() → PendingIntent
- AlarmManagerService
  - onCreate()
  - handleMessage()
  - send()
- NotificationManagerService
  - register PendingIntent
  - notify()
  - onClickHandler()
  - addNotification()
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that's registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.

Diagram:
- **DeferredDownload Activity**
  - OnCreate()
  - deferredDownload()
  - PendingIntent

- **Pending Intent**

- **ViewDownload Activity**
  - onCreate()
  - displayImage()

- **AlarmManager Service**
  - set()
  - handleMessage()

- **DeferredDownloadService**
  - onCreate()
  - onHandleIntent()
  - handleClickHandler()
  - addNotification()
  - notify()
Using Pending Intents in Deferred Download App

- DownloadActivity creates a PendingIntent that’s registered with the Alarm Service to start DeferredDownloadService to download an image in the future.
- DeferredDownloadService uses Notification Service to inform user when the image has been downloaded.

Diagram:
- DeferredDownloadActivity
  - Method: deferredDownload()
  - Method: onCreate()
- Pending Intent
- ViewDownloadActivity
  - Method: set()
  - Method: handleMessage()
  - Method: displayImage()
- AlarmManagerService
  - Method: send()
  - Method: onHandleIntent()
- NotificationManagerService
  - Method:_notify()
  - Method: addNotification()
  - Method: onClickHandler()
Programming DeferredDownloadActivity

• This Activity creates a PendingIntent & schedules it with Alarm Service

```java
public class DeferredDownloadActivity extends Activity {
    ...
    public void initiateDeferredDownload(View v) {
        Intent intent = new Intent(DownloadActivity.this,
                                     DeferredDownloadService.class);
        PendingIntent sender = PendingIntent.getService(
                                     DownloadActivity.this, 0,
                                     intent, 0);

        AlarmManager am =
        (AlarmManager) getSystemService(ALARM_SERVICE);

        am.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                downloadTime,
                sender);
    }
```

Create PendingIntent that starts a Service to download the image

Schedule an alarm to trigger the PendingIntent at the desired time

www.vogella.com/articles/AndroidNotifications/article.html#pendingintent has more
Programming DeferredDownloadService

- DeferredDownloadService uses the Notification Service to alert user when a requested image has been downloaded.

```java
public class DeferredDownloadService extends IntentService {
    ...
    protected void onHandleIntent(Intent intent) {
        String pathname = downloadImage(intent);

        Code to downloading image to pathname goes here

        Intent viewDownloadIntent =
            new Intent(this, ViewDownloadActivity.class);
        intent.setData(pathname);

        Prepare Intent to trigger if notification is selected

        PendingIntent contentIntent =
            PendingIntent.getActivity(this, 0, viewDownloadIntent, 0);

        Create PendingIntent to register with Notification Service
    }
}
```
Programming DeferredDownloadService

• DeferredDownloadService uses the Notification Service to alert user when a requested image has been downloaded

```java
public class DeferredDownloadService extends IntentService {
    ...
    protected void onHandleIntent(Intent intent) {
        ...
        Notification notification = new Notification.Builder(this)
            .setContentTitle("Image download complete")
            .setContentText(pathname).setSmallIcon(R.drawable.icon)
            .setContentIntent(contentIntent).build();

        NotificationManager nm = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);
        notification.flags |= Notification.FLAG_AUTO_CANCEL;
        notificationManager.notify(0, notification);
        ...
```
Programming ViewDownloadActivity

- This Activity is called when the user selects a notification that indicates the download has succeeded.

```java
public class ViewDownloadActivity extends Activity {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent callersIntent = getIntent();
        String pathname = callersIntent.getData().toString();
        displayImage(pathname);
    }
}
```

Get the pathname from the Intent that started this Activity

Display the image
Summary

• Pending Intents provide a powerful framework for an App to delegate some processing to another App at some future time or in some other context.
Summary

- Pending Intents provide a powerful framework for an App to delegate some processing to another App at some future time or in some other context.
- Pending Intents can also be used to communicate from a (Started) Service back to some other Android component.
- They are a bit complicated to learn/use…