



BROADCAST RECEIVER SERVICE

Broadcast receiver

- A broadcast receiver is a dormant component of the Android system.
- Only an *Intent* (for which it is registered) can bring it into action.
- Using a Broadcast Receiver, applications can register for a particular event. Once the event occurs, the system will notify all the registered applications.
 - Examples: Boot completed, Time tick
- The Broadcast Receiver's job is to activate some sw component, for example to notify the end user something occurred.

Registering a receiver

- There are two ways to register a Broadcast Receiver; one is Static and the other Dynamic.
- **Static:**
 - Use <receiver> tag in your Manifest file. (AndroidManifest.xml)
 - Not all events can be registered statically
 - Some events require permission
- **Dynamic:**
 - Use Context.registerReceiver () method to dynamically register an instance.
 - Note: Unregister when pausing

Type of broadcasts

- **Ordered Broadcasts:**

- These broadcasts are synchronous and follows the order specified using **android: priority attribute**.
- The **receivers with greater priority would receive the broadcast first**.

- **Normal Broadcasts:**

- Normal broadcasts are **not orderly**.

Simple example

- An activity creates a broadcast receiver that subscribes *dynamically* for TIME_TICK events (fired every minute)
- The receiver is registered to the event when the activity is started
- The receiver is unregistered when the hosting activity is paused.

Simple example

```
package com.example.bcastreceiverdemo;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends Activity {

    private final BroadcastReceiver timeBroadcastReceiver = new BroadcastReceiver(){}

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(timeBroadcastReceiver, new IntentFilter(Intent.ACTION_TIME_TICK));
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(timeBroadcastReceiver);
    }
}
```

Creates the receiver

Register the receiver
to receive time ticks...

Unregister the receiver
when paused

Simple example

```
private final BroadcastReceiver timeBroadcastReceiver = new BroadcastReceiver(){  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(MainActivity.this, "BroadCast Intent Receiver", Toast.LENGTH_SHORT).show();  
    }  
};
```

Good tutorial:

<http://www.grokkingandroid.com/android-tutorial-broadcastreceiver/>

Service

- A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use.
- A service is managed by android and has its own lifecycle
- For example, a broadcast receiver can use a service when something occurs.

Service type

- Intent Service
 - Simplest form of service
 - Created to execute a task in a separate thread and then exit
- Service
 - Started Service
 - Run until explicitly stopped (in the rare case android needs to kill it, the service will be restarted as soon as possible)
 - Started with startCommand method
 - Bound Service
 - Allows the exchange data with the interacting software component through an interface (set of methods)
 - Bind to a service interface

Intent Service: example



- The service needs to be registered in the manifest file
- The main activity creates an explicit intent pointing to the service
- The service is started and the *onHandleIntent* method executed
- Intents are queued and served serially

Intent Service: example

```
startService(new Intent(this,myIntentService.class));
```

```
<service android:name="myIntentService"/>
```

```
package com.example.servicedemo;

import android.app.IntentService;
import android.content.Intent;
import android.media.MediaPlayer;
import android.util.Log;

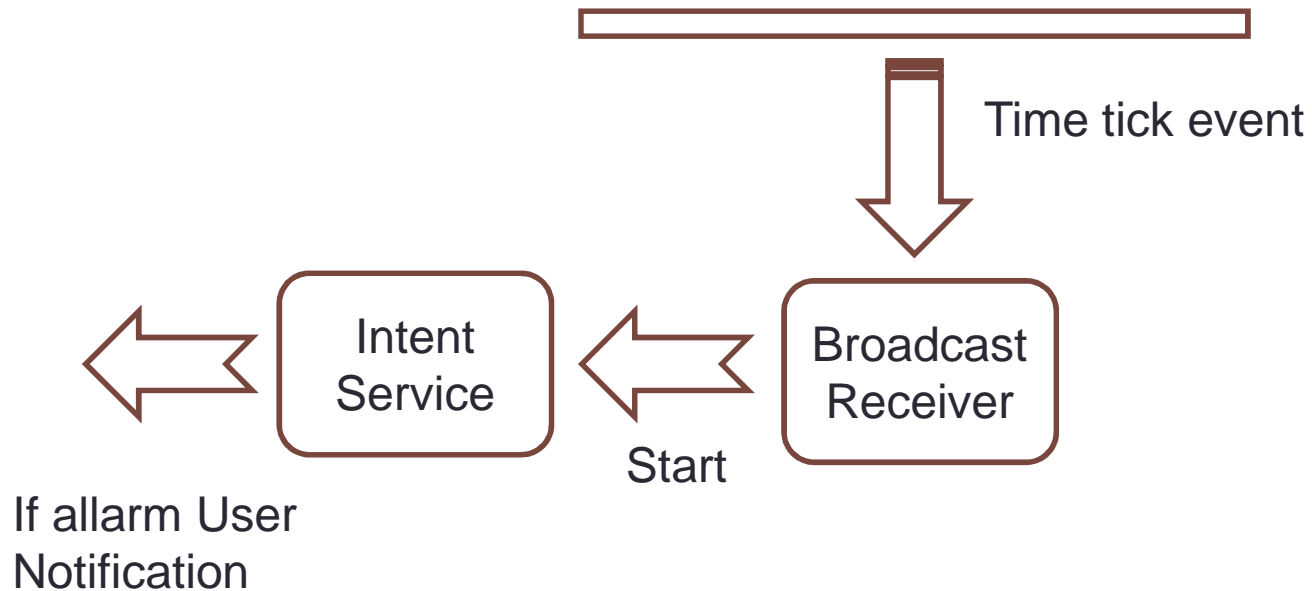
public class myIntentService extends IntentService{

    public myIntentService(){
        //name the worker thread, important only for debugging.
        super("myIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // TODO Auto-generated method stub
        Log.i("TEST","Intent Service...");
        MediaPlayer.create(this, R.raw.braincandy).start();
    }
}
```

Example

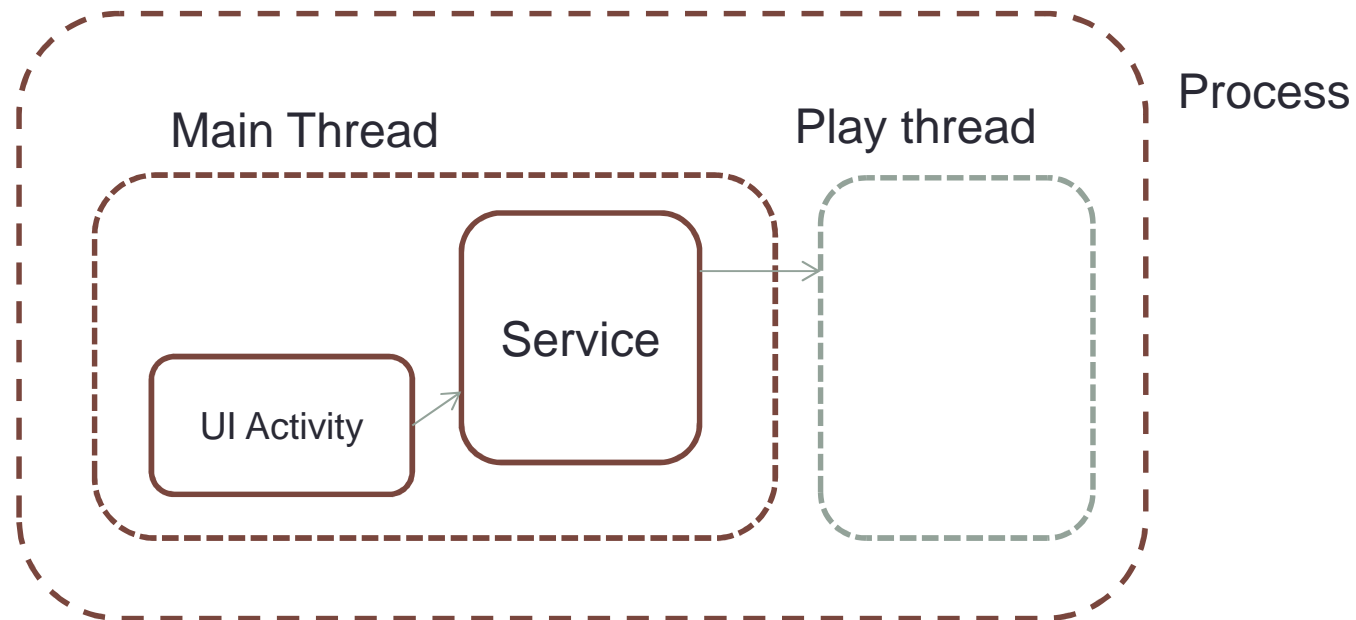
- Testing the weather condition periodically and send a notification if an alarm occurs



Started and bound service

- Any application component can use a service (even from a separate application), in the same way that any component can use an activity—by starting it with an Intent.
- To create a service, you must create a subclass of Service and override some callback methods that handle key aspects of the service lifecycle
 - Started services do not provide a programmatic interface to the client
 - Bound services do provide an interface

Started service: example, playing music



- An application that runs a player to play a song...
- The service is started from the Activity and then it spawns a thread

Example: playing music

```
package com.example.servicedemo;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

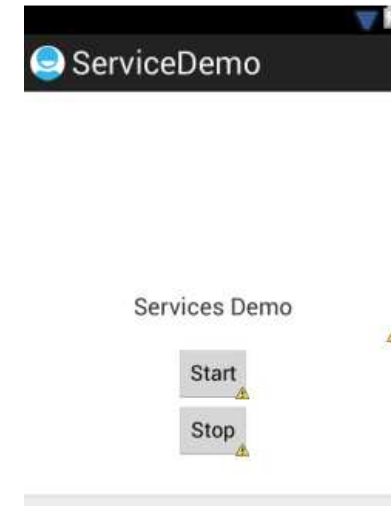
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        findViewById(R.id.buttonStart).setOnClickListener(this);
        findViewById(R.id.buttonStop).setOnClickListener(this);
    }

    public void onClick(View src) {

        try {
            Toast.makeText(this, src.getId(), Toast.LENGTH_LONG).show();
            switch (src.getId()) {
                case R.id.buttonStart:
                    startService(new Intent(MainActivity.this, MyService.class));
                    break;
                case R.id.buttonStop:
                    stopService(new Intent(MainActivity.this, MyService.class));
                    break;
            }
        } catch (Exception e) {
            Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
        }
    }
}
```



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="20dp"
        android:text="Services Demo"
        android:textSize="20sp" />

    <Button
        android:id="@+id/buttonStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start" >
    </Button>

    <Button
        android:id="@+id/buttonStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Stop" >
    </Button>

</LinearLayout>
```

Example: playing music

```
package com.example.servicedemo;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

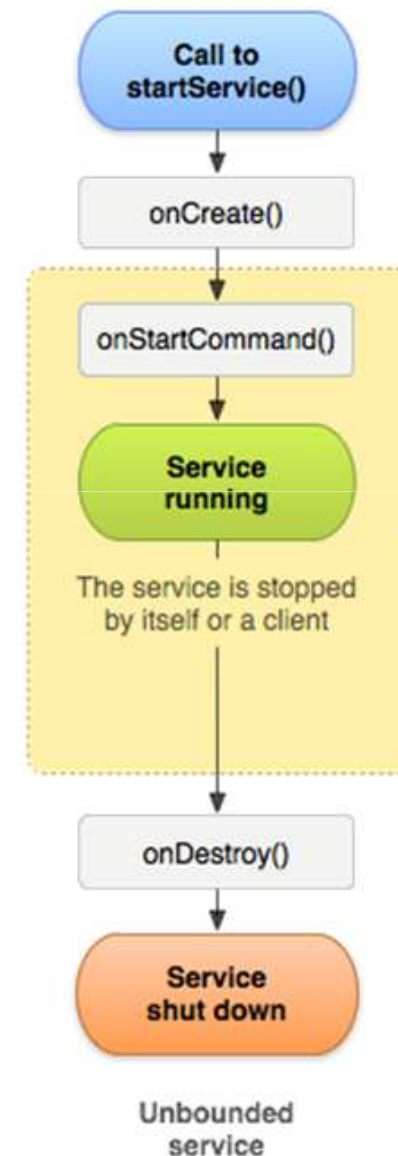
public class MyService extends Service {
    MediaPlayer player;

    @Override
    public void onCreate() {
        player = MediaPlayer.create(this, R.raw.braincandy);
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        new Thread(new Runnable(){
            public void run() {
                player.start();
            }
        }).start();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        player.stop();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

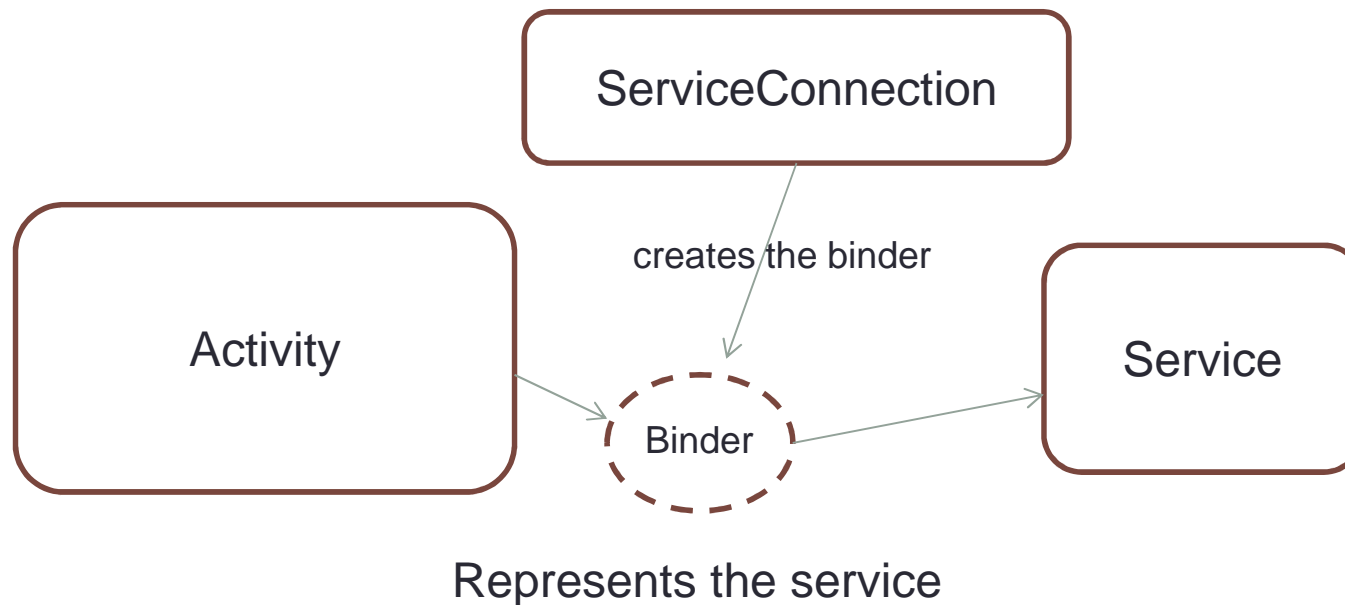


Main methods

- [onStartCommand\(\)](#)
 - The system calls this method when another component, such as an activity, requests that the service be started, by calling [startService\(\)](#).
- [onBind\(\)](#)
 - The system calls this method when another component wants to bind with the service (such as to perform RPC), by calling [bindService\(\)](#). Return null if no bounds are required

Bound Service – a short intro

- A service can be bounded to another SW component, meaning that it can invoke methods implemented by the service through a proxy (Binder) of the Service (which is seen as a remote object)
- Service connection is an interface monitoring connections to a service



Bound service

- To create a bound service, you must implement the [onBind\(\)](#) callback method to return an [IBinder](#) that defines the interface for communication with the service.
- Other application components can then call [bindService\(\)](#) to retrieve the interface and begin calling methods on the service.
 - The client can even call public methods defined in the service (see example)

Example

```
package com.example.boundservicedemo;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;

public class LocalBound extends Service{

    //Methods for the clients
    public String test() {return "hello from the service...";}
    public int testInt() {return 11;}

    //Implementation of the onBind Method,
    //returning a Binder object implementing the remote interface IBinder
    @Override
    public IBinder onBind(Intent arg0) {
        return new myLocalBinder();
    }

    //Binder class is an implementation of IBinder
    //that provides the standard support creating a local implementation of such an object.
    public class myLocalBinder extends Binder{
        LocalBound getService () {
            return LocalBound.this;
        }
    }
}
```

Example

-

```
package com.example.boundservicedemo;
```

```
import com.example.boundservicedemo.LocalBound.myLocalBinder;[]
```

```
public class MainActivity extends Activity {
```

```
    LocalBound myLocalBoundService;
```

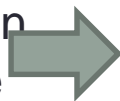
```
    public ServiceConnection myConnection = new ServiceConnection() {[]
```

```
        public void onCreate(Bundle savedInstanceState) {[]
```

```
        public void btnConn(View v) {[]
```

```
    }
```

Local
representation
of the remote
service



Interface: monitor the
state of the service



Example

Retrieve the
interface to the
service

```
package com.example.boundservicedemo;

import com.example.boundservicedemo.LocalBound.myLocalBinder;

public class MainActivity extends Activity {

    LocalBound myLocalBoundService;

    public ServiceConnection myConnection = new ServiceConnection() {

        public void onServiceConnected(ComponentName className, IBinder service) {
            myLocalBinder binder = (myLocalBinder)service;
            myLocalBoundService = binder.getService();
        }


        public void onServiceDisconnected(ComponentName className) {

        }

    };

    public void onCreate(Bundle savedInstanceState) {}

    public void btnConn(View v) {}
}
```



Example

```
package com.example.boundservicedemo;

+ import com.example.boundservicedemo.LocalBound.myLocalBinder;

public class MainActivity extends Activity {

    LocalBound myLocalBoundService;

    - public ServiceConnection myConnection = new ServiceConnection() {

        - @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            Intent intent = new Intent(this, LocalBound.class);
            bindService(intent, myConnection, Context.BIND_AUTO_CREATE);
        }

        - public void btnConn(View v) {
            String text = myLocalBoundService.test();
            ((TextView) findViewById(R.id.log)).setText(text);
        }
    }
}
```

automatically create the service as long as the binding exists.



System-level services

- The Android platform provides a lot of pre-defined services, usually exposed via a Manager class, see:
 - <http://developer.android.com/reference/android/content/Context.html>
- For example the next applications provides info about the currently connected network....

Example

```
package com.example.systemservicedemo;

import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Obtain a reference to the connectivity manager
        ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);

        //look for current active network (other methods available, see documentation)
        NetworkInfo ni = cm.getActiveNetworkInfo();

        //Write info to log
        Log.i("TEST",ni.toString());

        //write if roaming to a Toast
        if (ni.isRoaming()) {
            Toast.makeText(this,"roaming",1).show();
        }
        else
            Toast.makeText(this,"not roaming",1).show();
    }
}
```



Application	Tag	Text
com.example.systemservicedemo	TEST	NetworkInfo: type: WIFI[], state: CONNECTED/CONNECTED, reason: (unspecified), extra: (none), roaming: false, failover: false, isAvailable: true, isIpv4Connected: true, isIpv6Connected: false

Controlling Destroyed Service restart Options

`START_NOT_STICKY`

If the system kills the service after `onStartCommand()` returns, *do not* recreate the service, unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

`START_STICKY`

If the system kills the service after `onStartCommand()` returns, recreate the service and call `onStartCommand()`, but *do not* redeliver the last intent. Instead, the system calls `onStartCommand()` with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands, but running indefinitely and waiting for a job.

`START_REDELIVER_INTENT`

If the system kills the service after `onStartCommand()` returns, recreate the service and call `onStartCommand()` with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.

Service priority

- The system kills the process hosting a service if it is under heavy memory pressure.
- However, if this happens, the system will later try to restart the service (and a pending intent can be delivered again)
- A processes hosting service have higher priority than those running an activity

Example: use notification

- Send a message, displayed by the status bar
- Read the message associated to the notification

Example: UI

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff000066" ← Background color
    android:orientation="vertical" >

    <Button
        android:id="@+id/btnGo"
        android:layout_width="110px"
        android:layout_height="60px"
        android:layout_margin="10px"
        android:text=" Show " >
    </Button>

    <Button
        android:id="@+id/btnStop"
        android:layout_width="110px"
        android:layout_height="60px"
        android:layout_margin="10px"
        android:text=" Cancel " >
    </Button>

</LinearLayout>
```

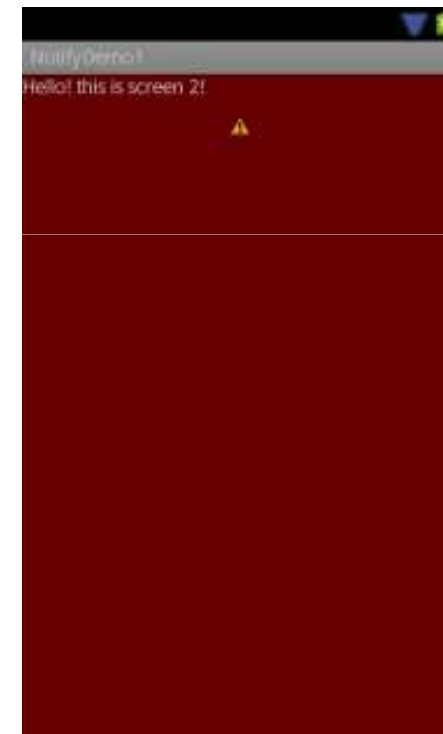


Example: UI

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/main2LinLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff660000"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/widget29"
        android:layout_width="251px"
        android:layout_height="69px"
        android:text="Hello! this is screen 2!" >
    </TextView>

</LinearLayout>
```



Example

```
package cis493.demos;

import android.app.Activity;

////////////////////////////////////////
public class NotifyDemo1 extends Activity {
    Button    btnGo;
    Button    btnStop;
    int       notificationId = 1;
    NotificationManager notificationManager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        btnGo = (Button)findViewById(R.id.btnGo);
        btnGo.setOnClickListener(new OnClickListener() {}

        btnStop = (Button)findViewById(R.id.btnStop);
        btnStop.setOnClickListener(new OnClickListener() {}

    } //onCreate
} //NotifyDemo1
```

Example: create a notification

```
btnGo = (Button)findViewById(R.id.btnGo);
btnGo.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        //define a notification manager
        String serName = Context.NOTIFICATION_SERVICE;
        notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

        //define notification using: icon, text, and timing.
        int icon = R.drawable.btn_star_big_on_selected;
        String tickerText = "1. My Notification TickerTapeText";
        long when = System.currentTimeMillis();
        Notification notification = new Notification(icon, tickerText, when);

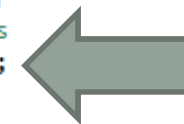
        //configure appearance of the notification
        String extendedTitle = "2. My Extended Title";
        String extendedText = "3. This is an extended and very important message";

        // set a Pending Activity to take care of the potential request the user
        // may have by clicking on the notification asking for more explanations
        Intent intent = new Intent(getApplicationContext(), NotifyHelper.class);
        intent.putExtra("extendedText", extendedText);
        intent.putExtra("extendedTitle", extendedTitle);

        //A PendingIntent describes an Intent and target action to perform with it.
        //Create a Pending Intent through getActivity factory method
        PendingIntent launchIntent =
            PendingIntent.getActivity(getApplicationContext(), 0, intent, 0);

        //Add Pending Intent to the notification
        notification.setLatestEventInfo(getApplicationContext(),
                                         extendedTitle, extendedText, launchIntent);

        //trigger notification
        notificationId = 1;
        notificationManager.notify(notificationId, notification);
    }
});
```



See next slide

Example: cancel a notification

```
btnStop = (Button)findViewById(R.id.btnStop);
btnStop.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        //canceling a notification
        notificationId = 1;
        notificationManager.cancel(notificationId);
    }
}
```

```
package cis493.demos;

import android.app.Activity;

public class NotifyHelper extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main2);
        Intent myData = getIntent();
        // show extra data sent to us
        String msg = "Extra data collected in the Notification intent's\n\n"
            + myData.getStringExtra("extendedTitle") + "\n"
            + myData.getStringExtra("extendedText");

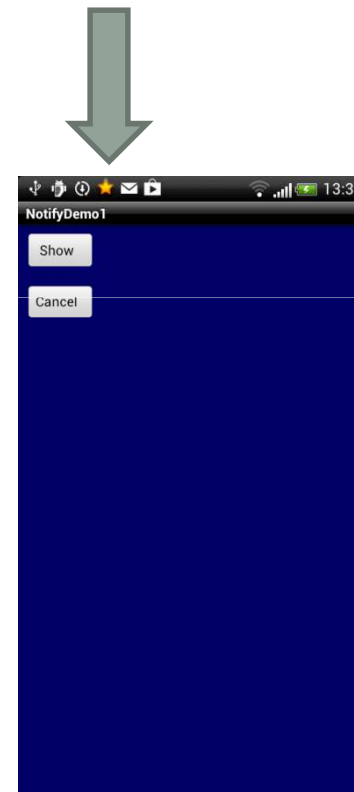
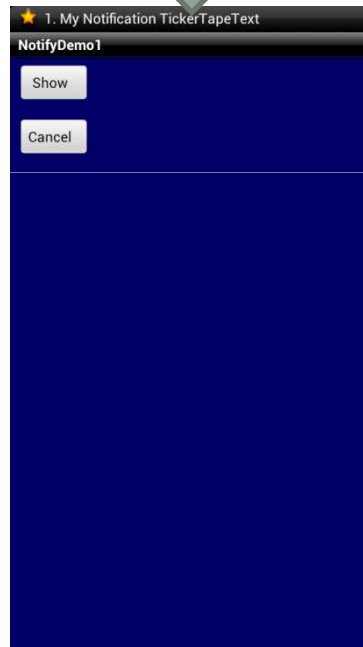
        Toast.makeText(getApplicationContext(), msg, 1).show();
    }
}
```

NotifyHelper

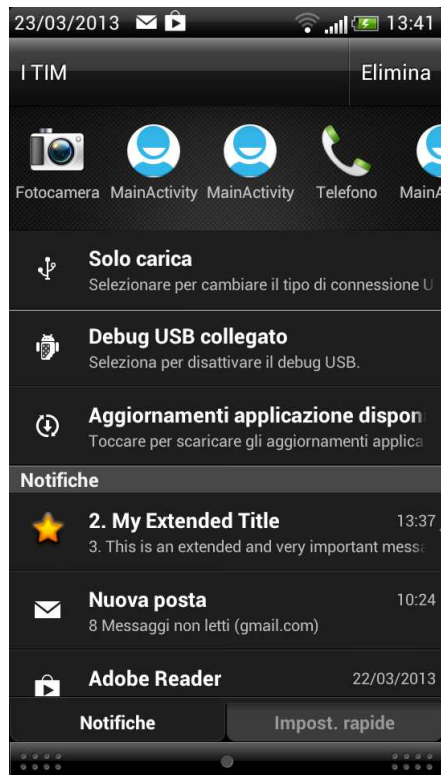
Running the application

Ticker Tape Text

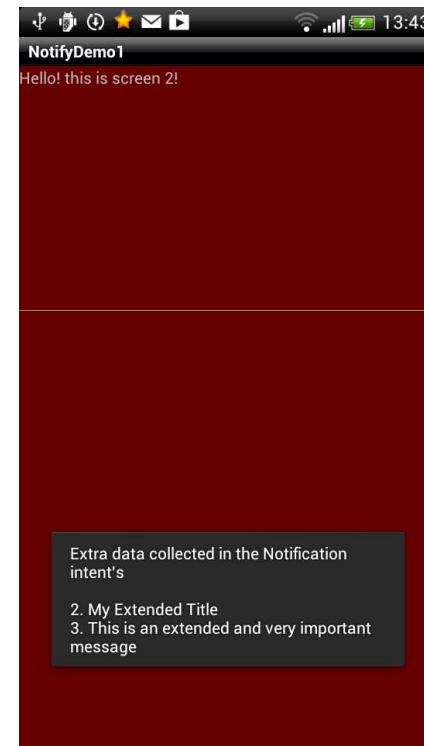
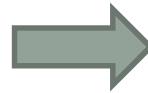
Icon (star) of the notification



Running the application



Activity launched
through intent



Extended
information

