Services

Marco Ronchetti
Università degli Studi di Trento

Service

An application component that can perform long-running operations in the background and does not provide a user interface.

So, what's different from a Thread?

a) Services are declared in the Manifest
b) Services can be exposed to other processes
c) Services do not need to be connected with an Activity

Service

A service can essentially take two forms: **Started** and **Bound**

Started

- A service is "started" when an application component (such as an activity) starts it by calling startService().
- Once started, a service can **run in the background indefinitely**, **even if the component that started it is destroyed**.
- Usually, a started service performs a single operation and **does not return a result to the caller**. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

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, get 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**.

Service

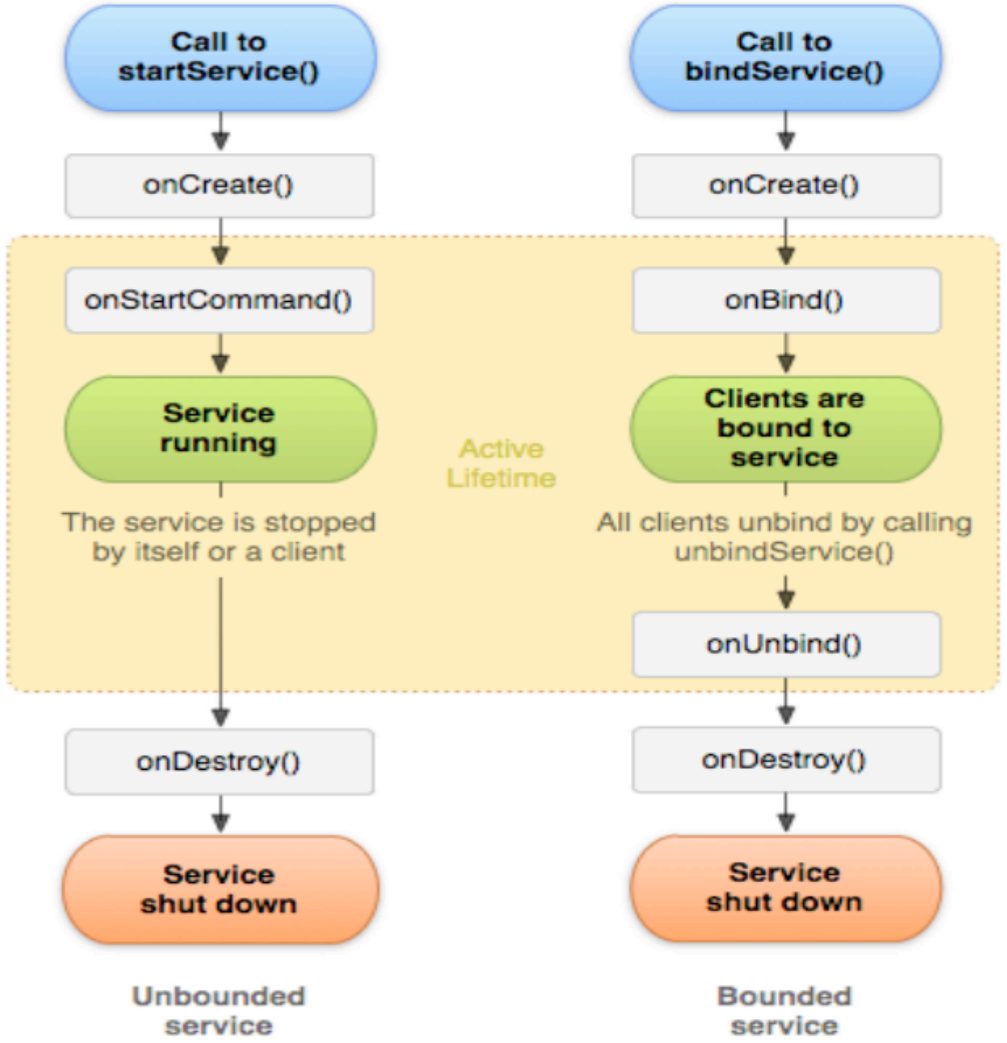
Although this documentation generally discusses these two types of services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding.

It's simply a matter of whether you implement a couple callback methods: onStartCommand() to allow components to start it and onBind() to allow binding.

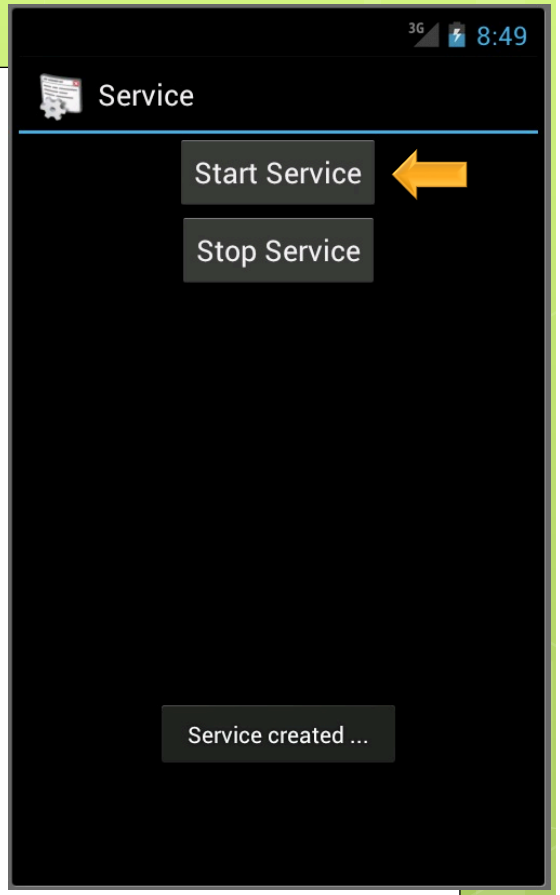
Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application), in the same way that any component can use an activity—by **starting it with an Intent**.

However, **you can declare the service as private**, in the manifest file, and block access from other applications.

Service lifecycle



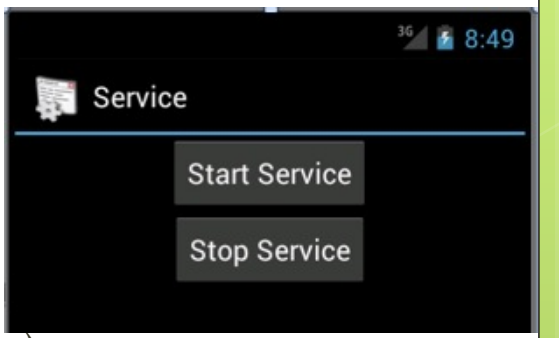
A simple service skeleton




```
import ...
public class SimpleService extends Service {
  public IBinder onBind(Intent arg0) {
        return null;
  }
  public void onCreate() {
        super.onCreate();
        Toast.makeText(this,"Service created ...", Toast.LENGTH_LONG).show();
  }
  public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service destroyed ...", Toast.LENGTH_LONG).show();
  }
}
```

Adapted from saigeethamn.blogspot.it

Using our simple service




```
import ...
public class SimpleServiceController extends Activity {
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Button start = (Button)findViewById(R.id.serviceButton);
      Button stop = (Button)findViewById(R.id.cancelButton);
      start.setOnClickListener(startListener);
      stop.setOnClickListener(stopListener);
  }
  private OnClickListener startListener = new OnClickListener() {
      public void onClick(View v){
            startService(new Intent(SimpleServiceController.this,SimpleService.class));
   }};
   private OnClickListener stopListener = new OnClickListener() {
       public void onClick(View v){
            stopService(new Intent(SimpleServiceController.this,SimpleService.class));
    }};
}
```

Adapted from saigeethamn.blogspot.it

Service methods and IVs

int **mStartMode**; // indicates how to behave if the service is killed
IBinder **mBinder**; // interface for clients that bind
boolean **mAllowRebind**; // indicates whether onRebind should be used

public void **onCreate()**
- The service is being created

public int **onStartCommand(Intent intent, int flags, int startId)** {
- The service is starting, due to a call to startService()

public IBinder **onBind(Intent intent)** {
- A client is binding to the service with bindService()

public boolean **onUnbind(Intent intent)** {
- All clients have unbound with unbindService()

public void **onRebind(Intent intent)** {
- A client is binding to the service with bindService() after onUnbind() has been called

public void **onDestroy()** {
- The service is no longer used and is being destroyed

IntentService

a subclass for Services that handle asynchronous requests (expressed as Intents) on demand.

Clients send requests through startService(Intent) calls; the service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work.

"work queue processor" pattern

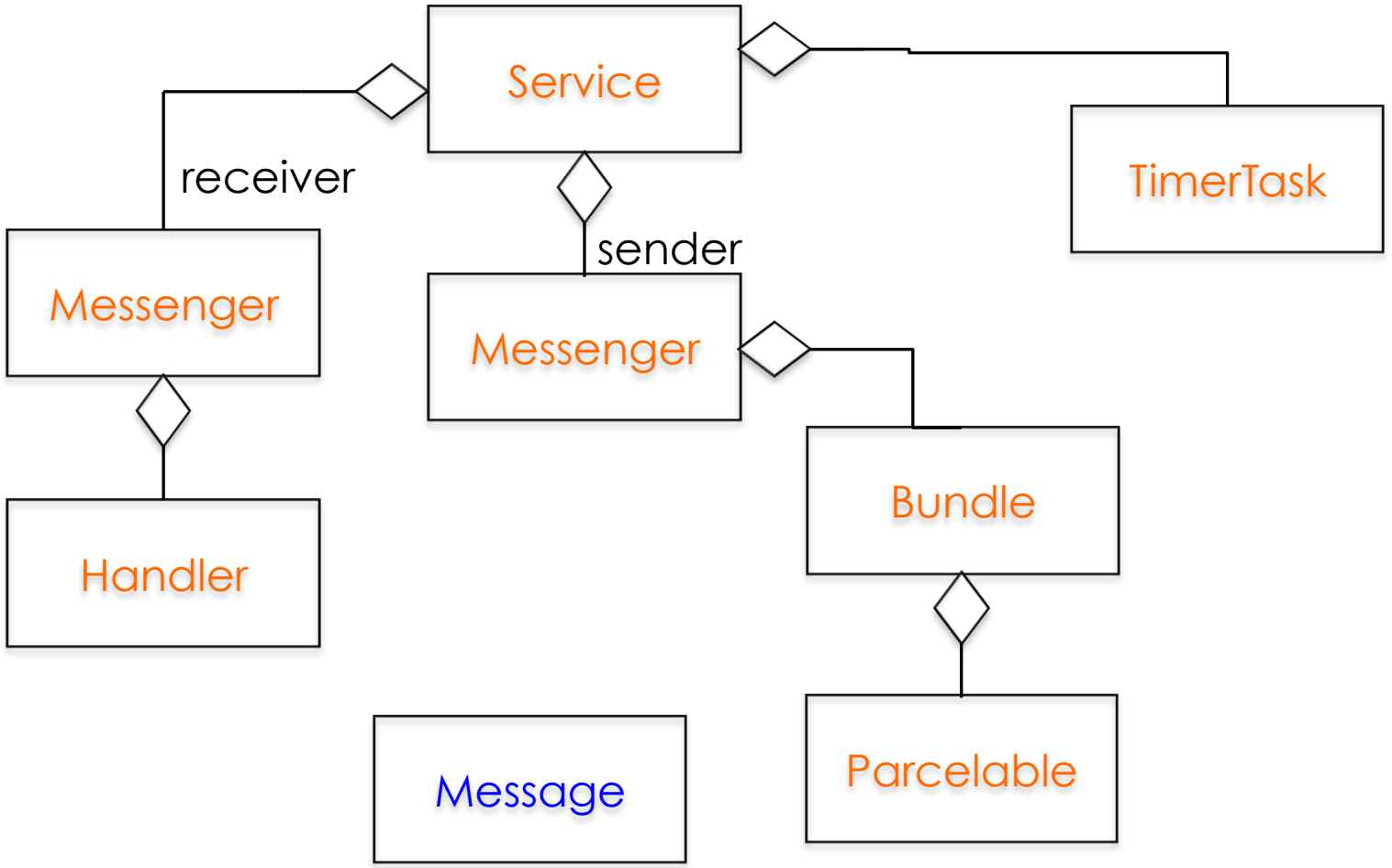
To use it, extend IntentService and implement onHandleIntent(Intent). IntentService will receive the Intents, launch a worker thread, and stop the service as appropriate.

All requests are handled on a single worker thread -- they may take as long as necessary (and will not block the application's main loop), but only one request will be processed at a time.

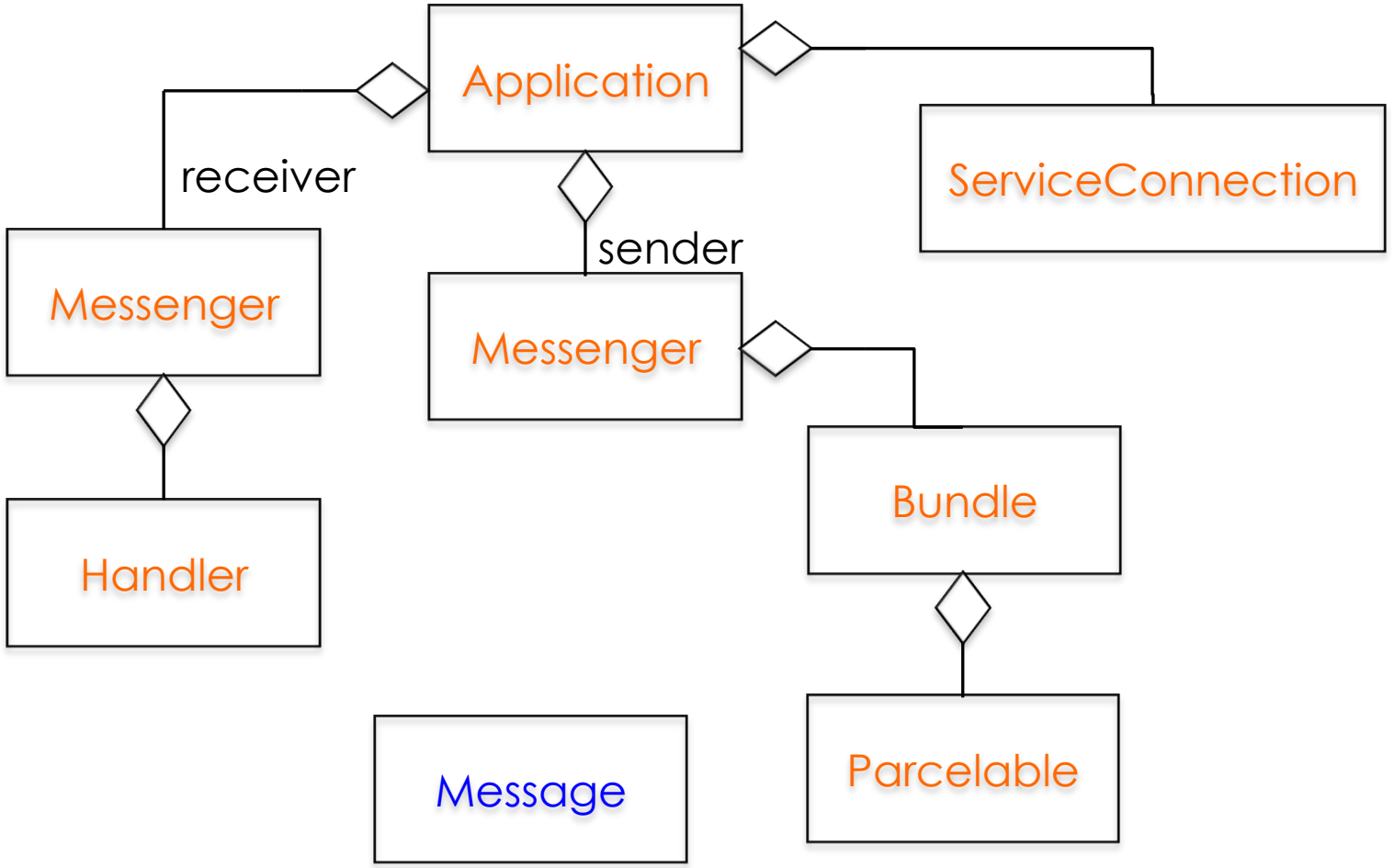
Architecture and introduction and essential classes

Marco Ronchetti
Università degli Studi di Trento

Service structure



Client structure



Binder

Base class for a remotable object.

the core part of a lightweight remote procedure call mechanism defined by the interface IBinder.

This class is an implementation of IBinder that provides the standard support creating a local implementation of such an object.

Bundle

A mapping from String values to various Parcelable types.

Parcel

A Parcel is a serialized object. It can contain both flattened data that will be unflattened on the other side of the IPC, and references to live iBinder objects that will result in the other side receiving a proxy IBinder connected with the original IBinder in the Parcel.

Parcel is **not** a general-purpose serialization mechanism. This class (and the corresponding Parcelable API for placing arbitrary objects into a Parcel) is designed as a high-performance IPC transport.

Handler

allows you to send and process Message and Runnable objects associated with a thread's MessageQueue.

Each Handler instance is associated with a single thread and that thread's message queue.

When you create a new Handler, it is bound to the thread/ message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

Handlers are used to

(1) to schedule messages and runnables to be executed as some point in the future;

(2) to enqueue an action to be performed on a different thread than your own.

void handleMessage(Message msg)

Message

Defines a message containing a description and arbitrary data object that can be sent to a Handler.

It contains two extra int fields and an extra object field that allow you to not do allocations in many cases.

To create one, it's best to use a factory method: Message.obtain()

Fields

Object obj

- An arbitrary object to send to the recipient.

Messenger replyTo

- Optional Messenger where replies to this message can be sent.

int what

- User-defined message code so that the recipient can identify what this message is about.

setData(Bundle b)

Bundle: a type of Parcel,

- Sets a Bundle of arbitrary data values.

Message

public int arg1
public int arg2
arg1 and arg2 are lower-cost alternatives to using setData() if you only need to store a few integer values.

public Object obj
An arbitrary object to send to the recipient.

public Messenger replyTo
Optional Messenger where replies to this message can be sent.

public int sendingUid
Optional field indicating the uid that sent the message.

public int what
User-defined message code so that the recipient can identify what this message is about.

Message

static Message obtain()
Return a new Message instance from the global pool.

Same as obtain(), but sets the value for the come of the instance variables:.
static Message obtain(Handler h)
static Message obtain(Handler h, int what)
static Message obtain(Handler h, int what, Object obj)
static Message obtain(Handler h, int what, int arg1, int arg2)
static Message obtain(Handler h, int what, int arg1, int arg2, Object obj)

static Message obtain(Handler h, Runnable callback)
Same as obtain(android.os.Handler), but assigns a callback Runnable on the Message that is returned.

static Message obtain(Message orig)
Same as obtain(), but copies the values of an existing message (including its target) into the new one.

Messenger

Reference to a Handler, which others can use to send messages to it. This allows for the implementation of message-based communication across processes, by creating a Messenger pointing to a Handler in one process, and handing that Messenger to another process.

ServiceConnection

Interface for monitoring the state of an application service.

void onServiceConnected(ComponentName name, IBinder service)

- Called when a connection to the Service has been established, with the IBinder of the communication channel to the Service.

void onServiceDisconnected(ComponentName name)

- Called when a connection to the Service has been lost.

java.util.Timer and TimerTask

Timer

facility for threads to schedule tasks for future execution in a background thread. Tasks may be scheduled for one-time execution, or for repeated execution at regular intervals.

schedule(TimerTask task, Date time)

scheduleAtFixedRate(TimerTask task, Date firstTime, long period)

TimerTask

A task that can be scheduled for one-time or repeated execution by a Timer.

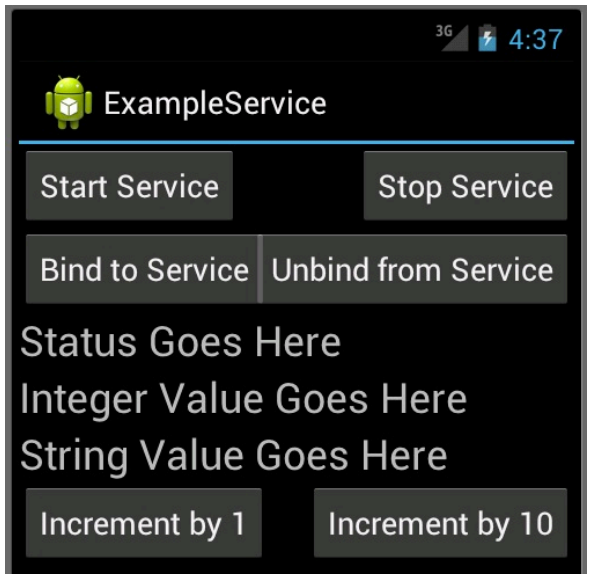
public abstract void run()

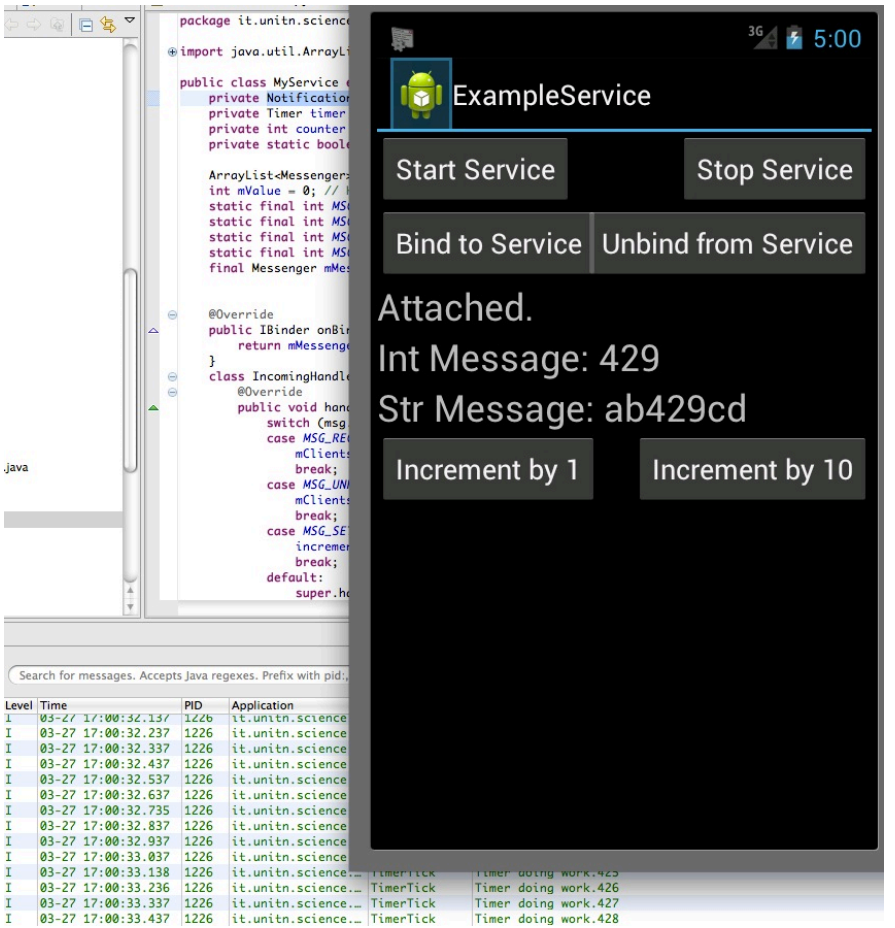
A full example part 1: activity implementation

Marco Ronchetti
Università degli Studi di Trento

A full example

Get the src files from
http://latemar.science.unitn.it/segue_userFiles/2018Mobile/ServiceFullDemoSrc.zip





Code adapted from an example on StackOverflow

Protocol

```
Class Protocol {
    static final int MSG_REGISTER_CLIENT = 1;
    static final int MSG_UNREGISTER_CLIENT = 2;
    static final int MSG_SET_INT_VALUE = 3;
    static final int MSG_SET_STRING_VALUE = 4;
}
```

Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exampleservice"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon"
          android:label="@string/app_name">
    <activity android:name=".MainActivity"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  <service android:name=".MyService"></service>
  </application>
  <uses-sdk android:minSdkVersion="13" />
</manifest>
```

Running service in processes

You can specify a process name with a colon in front

```
<service android:name=".MyService"
android:process=:myprocessname ></service>
```

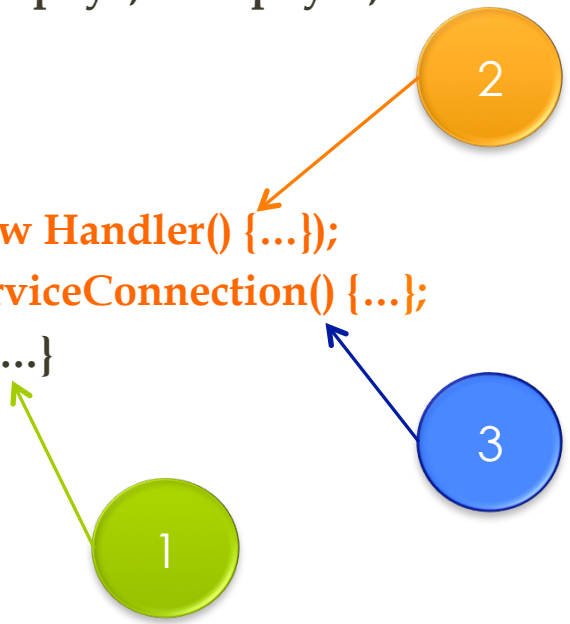
Your service will then run as a different process - thus in a different thread.

You can set this attribute so that each component runs in its own process or so that some components share a process while others do not.

You can also set it so that components of different applications run in the same process—provided that the applications share the same Linux user ID and are signed with the same certificates.

ServiceFullDemoActivity

```
public class ServiceFullDemoActivity extends Activity {
    Button btnStart, btnStop, btnBind, btnUnbind, btnUpby1, btnUpby10;
    TextView textStatus, textIntValue, textStrValue;
    Messenger mService = null;
    boolean mIsBound;
    final Messenger mMessenger = new Messenger(new Handler() {...});
    private ServiceConnection mConnection = new ServiceConnection() {...};
    public void onCreate(Bundle savedInstanceState) {...}
    protected void onDestroy() {...}
}
```



OnCreate

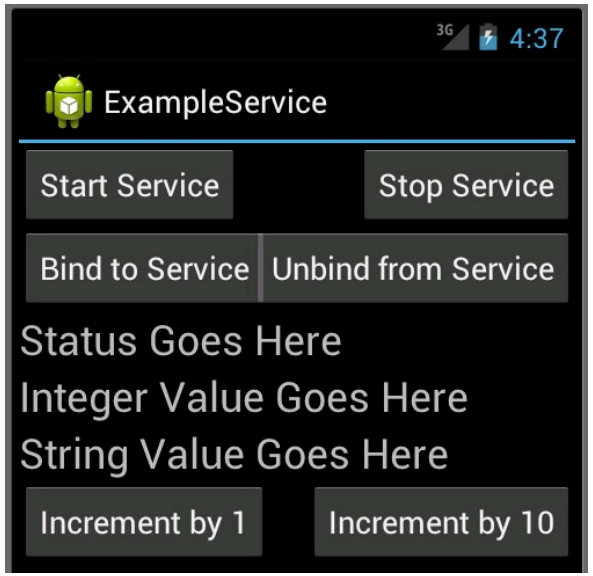
1

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    btnStart = (Button)findViewById(R.id.btnStart);
    btnStop = (Button)findViewById(R.id.btnStop);
    btnBind = (Button)findViewById(R.id.btnBind);
    btnUnbind = (Button)findViewById(R.id.btnUnbind);
    textStatus = (TextView)findViewById(R.id.textStatus);
    textIntValue = (TextView)findViewById(R.id.textIntValue);
    textStrValue = (TextView)findViewById(R.id.textStrValue);
    btnUpby1 = (Button)findViewById(R.id.btnUpby1);
    btnUpby10 = (Button)findViewById(R.id.btnUpby10);

    btnStart.setOnClickListener(btnStartListener);
    btnStop.setOnClickListener(btnStopListener);
    btnBind.setOnClickListener(btnBindListener);
    btnUnbind.setOnClickListener(btnUnbindListener);
    btnUpby1.setOnClickListener(btnUpby1Listener);
    btnUpby10.setOnClickListener(btnUpby10Listener);

    restoreMe(savedInstanceState);

  //If the service is running when the activity starts, we want to automatically bind to it.
    if (MyService.isRunning()) {
       doBindService() ;
}
```



Listeners

1

```
private OnClickListener btnStartListener = new OnClickListener() {
    public void onClick(View v){
        startService(new Intent(ServiceFullDemoActivity.this, MyService.class));
    }
};
private OnClickListener btnStopListener = new OnClickListener() {
    public void onClick(View v){
        doUnbindService();
        stopService(new Intent(ServiceFullDemoActivity.this, MyService.class));
    }
};
private OnClickListener btnBindListener = new OnClickListener() {
    public void onClick(View v){
        doBindService();
    }
};
private OnClickListener btnUnbindListener = new OnClickListener() {
    public void onClick(View v){
        doUnbindService();
    }
};
```

Listeners

1

```
private OnClickListener btnUpby1Listener = new OnClickListener() {
    public void onClick(View v){
        sendMessageToService(1);
    }
};
private OnClickListener btnUpby10Listener = new OnClickListener() {
    public void onClick(View v){
        sendMessageToService(10);
    }
};
private void sendMessageToService(int intvaluetosend) {
    if (mIsBound) {
        if (mService != null) {
            try {
                Message msg = Message.obtain(null, Protocol.MSG_SET_INT_VALUE, intvaluetosend, 0);
                msg.replyTo = mMessenger;
                mService.send(msg);
            } catch (RemoteException e) {
            }
        }
    }
}
```

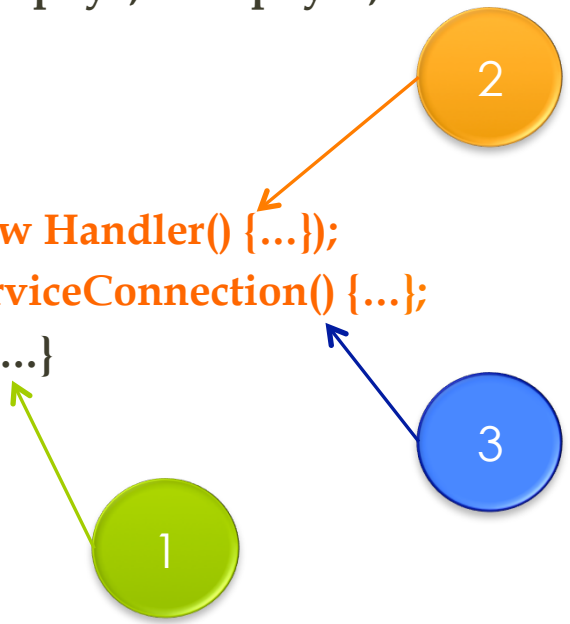
onSaveInstanceState

1

```
@Override
  protected void onSaveInstanceState(Bundle outState) {
     super.onSaveInstanceState(outState);
     outState.putString("textStatus", textStatus.getText().toString());
     outState.putString("textIntValue", textIntValue.getText().toString());
     outState.putString("textStrValue", textStrValue.getText().toString());
  }
  private void restoreMe(Bundle state) {
     if (state!=null) {
        textStatus.setText(state.getString("textStatus"));
        textIntValue.setText(state.getString("textIntValue"));
        textStrValue.setText(state.getString("textStrValue"));
     }
  }
```

ServiceFullDemoActivity

```
public class ServiceFullDemoActivity extends Activity {
    Button btnStart, btnStop, btnBind, btnUnbind, btnUpby1, btnUpby10;
    TextView textStatus, textIntValue, textStrValue;
    Messenger mService = null;
    boolean mIsBound;
    final Messenger mMessenger = new Messenger(new Handler() {...});
    private ServiceConnection mConnection = new ServiceConnection() {...};
    public void onCreate(Bundle savedInstanceState) {...}
    protected void onDestroy() {...}
}
```

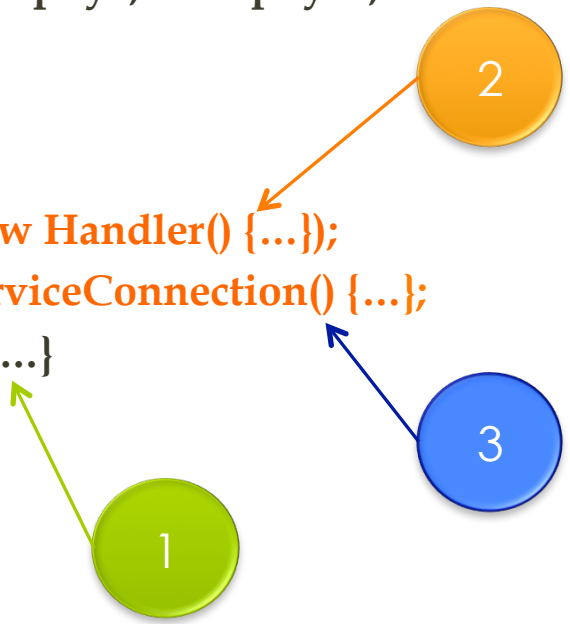


```
final Messenger mMessenger = new Messenger(new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case Protocol.MSG_SET_INT_VALUE:
            textIntValue.setText("Int Message: " + msg.arg1);
            break;
        case Protocol.MSG_SET_STRING_VALUE:
            String str1 = msg.getData().getString("str1");
            textStrValue.setText("Str Message: " + str1);
            break;
        default:
            super.handleMessage(msg);
        }
    }
});
```

2

ServiceFullDemoActivity

```
public class ServiceFullDemoActivity extends Activity {
    Button btnStart, btnStop, btnBind, btnUnbind, btnUpby1, btnUpby10;
    TextView textStatus, textIntValue, textStrValue;
    Messenger mService = null;
    boolean mIsBound;
    final Messenger mMessenger = new Messenger(new Handler() {...});
    private ServiceConnection mConnection = new ServiceConnection() {...};
    public void onCreate(Bundle savedInstanceState) {...}
    protected void onDestroy() {...}
}
```



Create ServiceConnection

3

```
private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder service) {
        mService = new Messenger(service);
        textStatus.setText("Attached.");
        try {
            Message msg = Message.obtain(null, Protocol.MSG_REGISTER_CLIENT);
            msg.replyTo = mMessenger;
            mService.send(msg);
        } catch (RemoteException e) {
            // In this case the service has crashed before we could even do anything with it
        }
    }

    public void onServiceDisconnected(ComponentName className) {
        // This is called when the connection with the service has been
        // unexpectedly disconnected - process crashed.
        mService = null;
        textStatus.setText("Disconnected.");
    }
};
```

bind - unbind

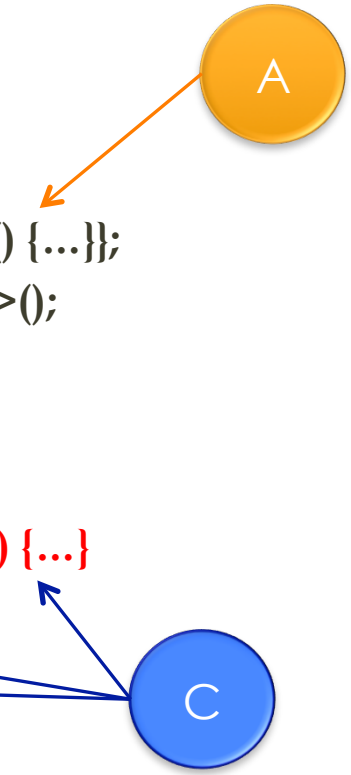
```
void doBindService() {
    bindService(new Intent(this, MyService.class), mConnection, Context.BIND_AUTO_CREATE);
     mIsBound = true;
     textStatus.setText("Binding.");
}
 void doUnbindService() {
       if (mIsBound) {
          // If we have received the service, and registered with it, then now is the time to unregister.
          if (mService != null) {
             try {
                Message msg = Message.obtain(null, Protocol.MSG_UNREGISTER_CLIENT);
                msg.replyTo = mMessenger;
                mService.send(msg);
             } catch (RemoteException e) {// nothing special to do if the service has crashed.
             }
          }
          // Detach our existing connection.
          unbindService(mConnection);
          mIsBound = false;
          textStatus.setText("Unbinding.");
       }
    }
```

A full example part 2: service implementation

Marco Ronchetti
Università degli Studi di Trento

MyService

```
public class MyService extends Service {
   private Timer timer = new Timer();
   private int counter = 0, incrementby = 1;
   final Messenger mMessenger = new Messenger(new Handler() {...});
   ArrayList<Messenger> mClients = new ArrayList<Messenger>();
   private static boolean isRunning = false;
   public static boolean isRunning() {return isRunning; }
    public void onCreate() {...}
   public int onStartCommand(Intent intent, int flags, int startId) {...}
    public IBinder onBind(Intent intent){...}
   public void onDestroy() {...}
}
```



Create Messenger – MyService

```
final Messenger mMessenger = new Messenger(new Handler() {
    // Handler of incoming messages from clients.
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case Protocol.MSG_REGISTER_CLIENT:
            mClients.add(msg.replyTo);
            break;
        case  Protocol.MSG_UNREGISTER_CLIENT:
            mClients.remove(msg.replyTo);
            break;
        case Protocol.MSG_SET_INT_VALUE:
            incrementby = msg.arg1;
            break;
        default:
            super.handleMessage(msg);
        }
    }
});
```

A

onCreate - MyService

B

```
public void onCreate() {
    super.onCreate();
    Log.i("MyService", "Service Created.");
    Toast.makeText(this,"Service Created", Toast.LENGTH_LONG).show();
    timer.scheduleAtFixedRate(new TimerTask(){ public void run() {
      Log.i("TimerTick", "Timer doing work." + counter);
       try {
          counter += incrementby;
          sendMessageToUI(counter);
       } catch (Throwable t) {Log.e("TimerTick", "Timer Tick Failed.", t); }
    }}, 0, 100L);
    isRunning = true;
  }
```

sendMessageToUI - onCreate - MyService

B

```
private void sendMessageToUI(int intvaluetosend) {
    for (int i=mClients.size()-1; i>=0; i--) {
        try {
            // Send data as an Integer
            mClients.get(i).send(Message.obtain(null,
                                    Protocol.MSG_SET_INT_VALUE, intvaluetosend, 0));
            //Send data as a String
            Bundle b = new Bundle();
            b.putString("str1", "ab" + intvaluetosend + "cd");
            Message msg = Message.obtain(null, MSG_SET_STRING_VALUE);
            msg.setData(b);
            mClients.get(i).send(msg);
        } catch (RemoteException e) {
            // The client is dead. Remove it from the list;
            //we are going through the list from back to front so this is safe
            //to do inside the loop.
            mClients.remove(i);
        }
    }
}
```

onStartCommand – onDestroy - My Service

```
public int onStartCommand(Intent intent, int flags, int startId) {
    Log.i("MyService", "Received start id " + startId + ": " + intent);
    Toast.makeText(this,"Service started"+ startId ,
                                    Toast.LENGTH_LONG).show();
    return START_STICKY; // run until explicitly stopped.
  }

public void onDestroy() {
    super.onDestroy();
    if (timer != null) {timer.cancel();}
    counter=0;
   Log.i("MyService", "Service Stopped.");
   Toast.makeText(this,"Service Stopped ", Toast.LENGTH_LONG).show();
    isRunning = false;
  }

public IBinder onBind(Intent intent) {
    return mMessenger.getBinder();
  }
```