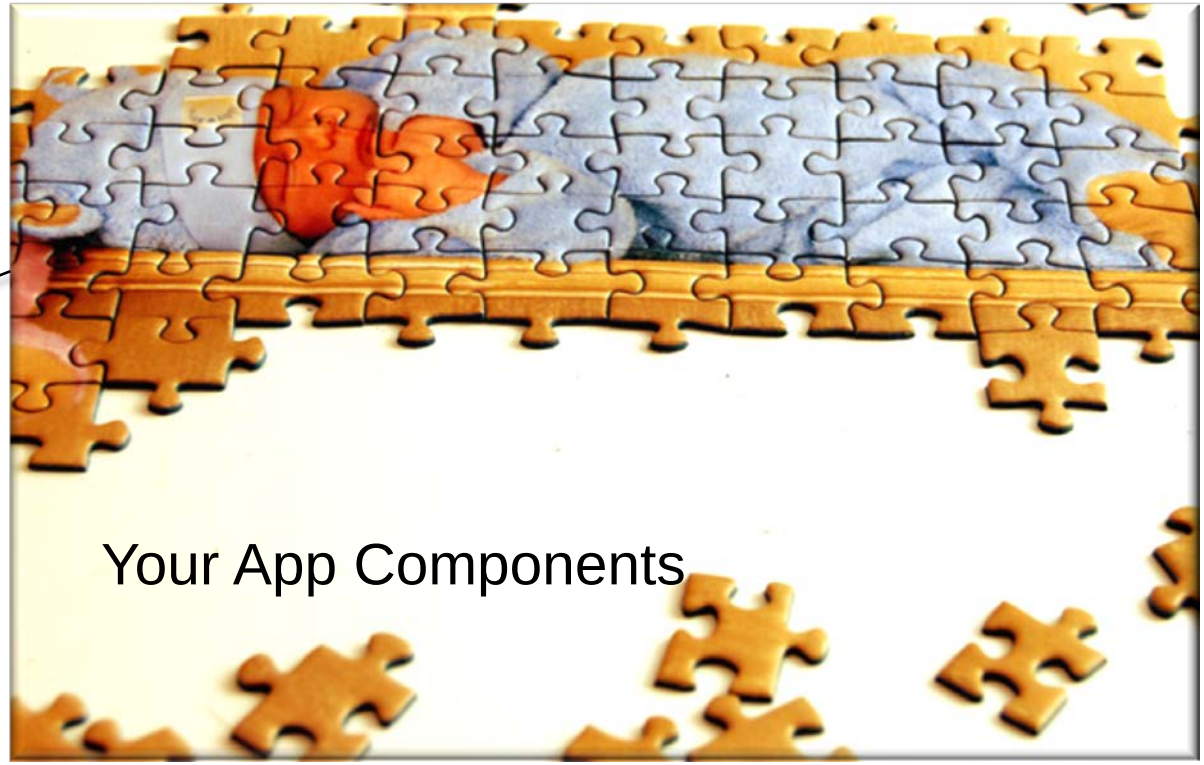
Integrating a Service into an Application

Integrating a Service into an Application

- The Android Service framework provides a semi-complete portion of an Application

Android



The manifest file tells Android how the Activity & Service components "plug-in" to the framework

See developer.android.com/guide/topics/manifest/manifest-intro.html

Integrating a Service into an Application

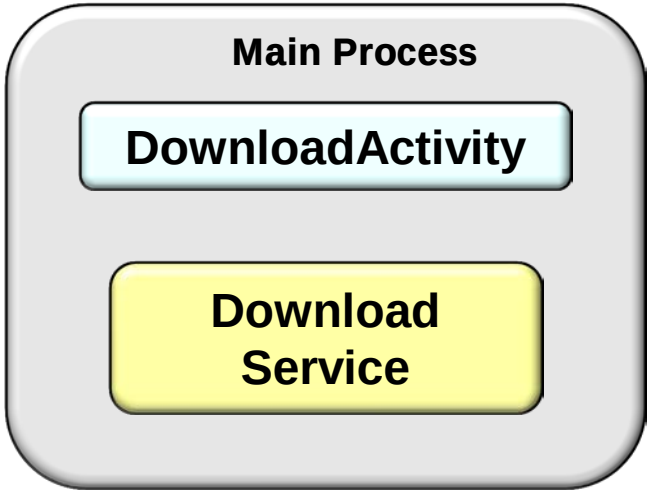
- The Android Service framework provides a semi-complete portion of an Application
- Include the Service in the AndroidManifest.xml file

```
<service
    android:enabled=
        ["true" | "false"]
    android:exported=
        ["true" | "false"]
    android:icon=
        "drawable resource"
    android:isolatedProcess=
        ["true" | "false"]
    android:label=
        "string resource"
    android:name="string"
    android:permission="string"
    android:process="string" >
    ...
</service>
```

See developer.android.com/guide/topics/manifest/service-element.html

Integrating a Service into an Application

- The Android Service framework provides a semi-complete portion of an Application
- Include the Service in the AndroidManifest.xml file
  - Add <service> element as a child of <application> element

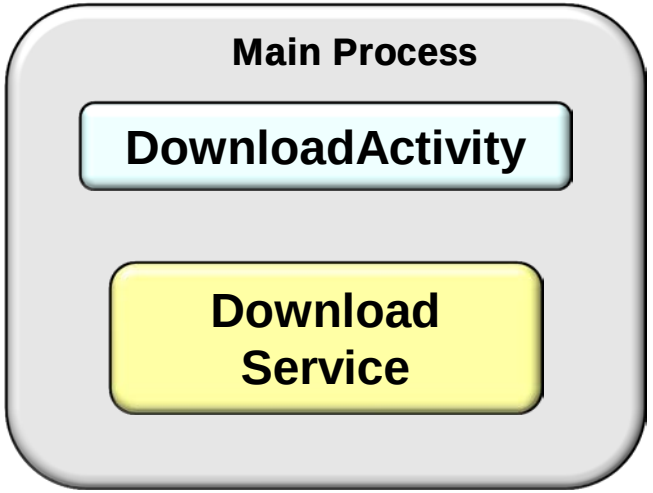


AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"/>
...
```

Integrating a Service into an Application

- The Android Service framework provides a semi-complete portion of an Application
- Include the Service in the AndroidManifest.xml file
  - Add <service> element as a child of <application> element
  - Provide the android:name to reference the Service class



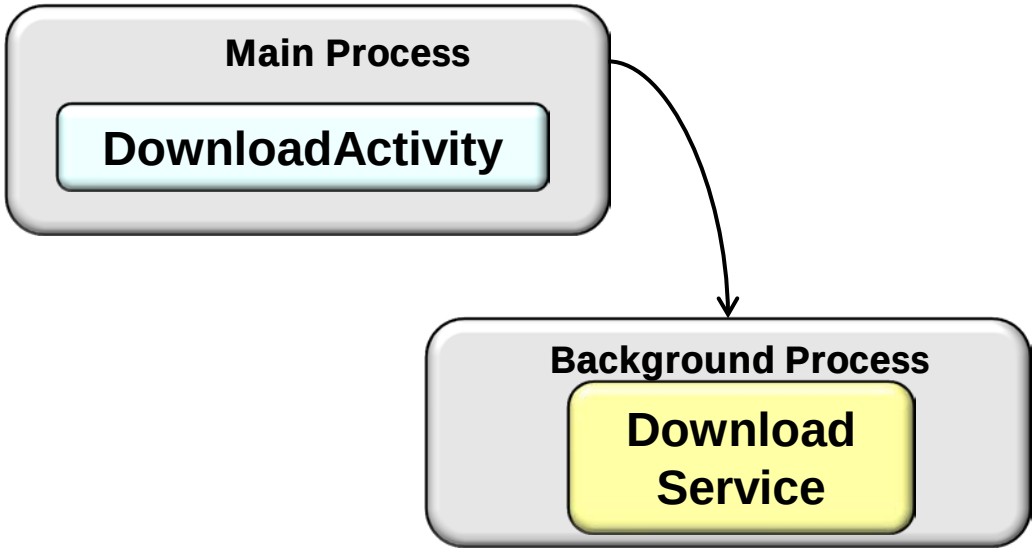
AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"/>
...
```

Services do not automatically run in their own processes or threads

Integrating a Service into an Application

- The Android Service framework provides a semi-complete portion of an Application
- Include the Service in the AndroidManifest.xml file
  - Add <service> element as a child of <application> element
  - Provide the android:name to reference the Service class
  - Use android:process= ":myProcess" to run a Service in a separate process



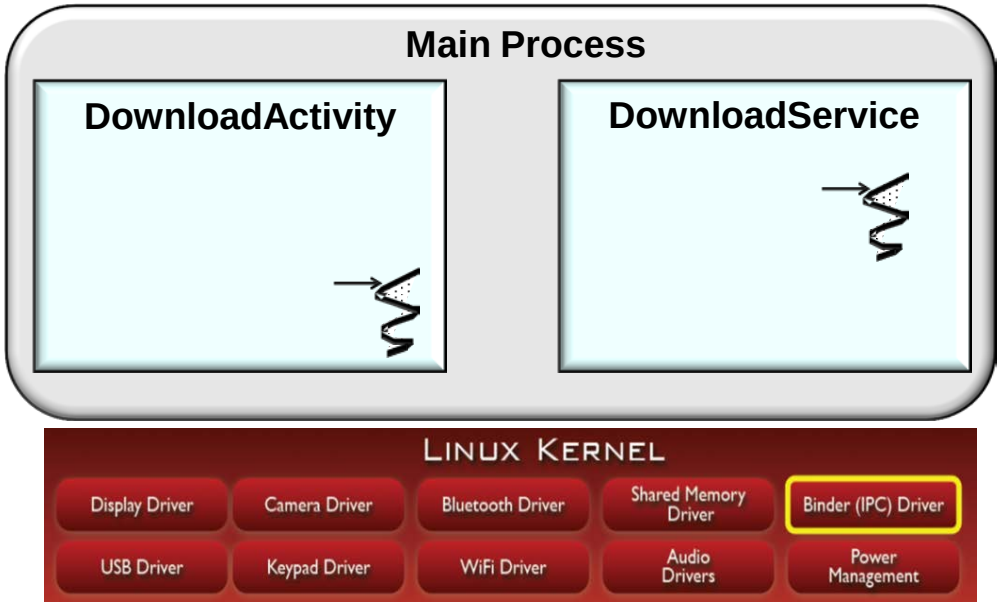
AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"
         android:process=
         ":myProcess"/>
...
```

Overview of Service Deployment Models

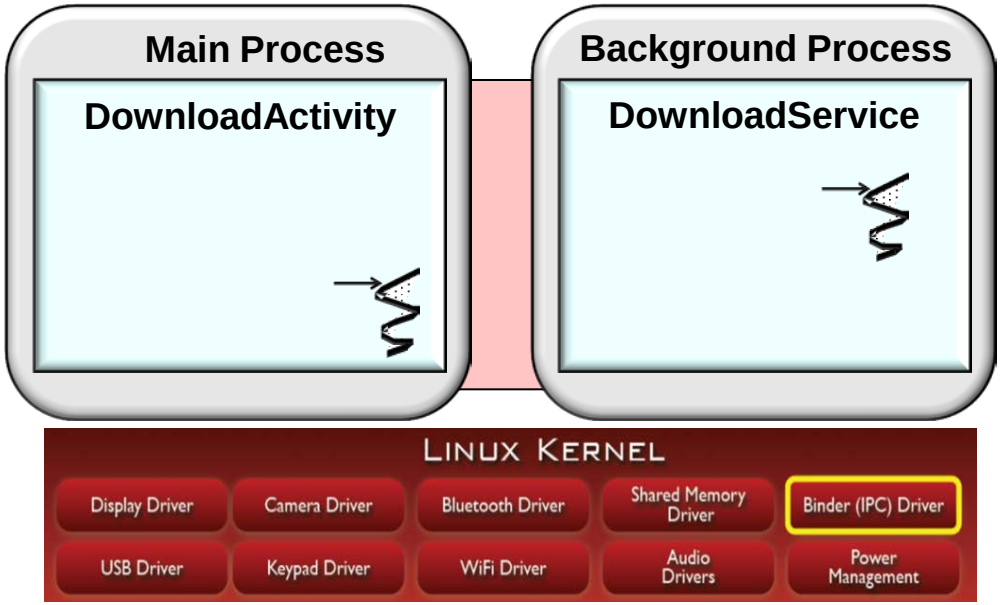
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients



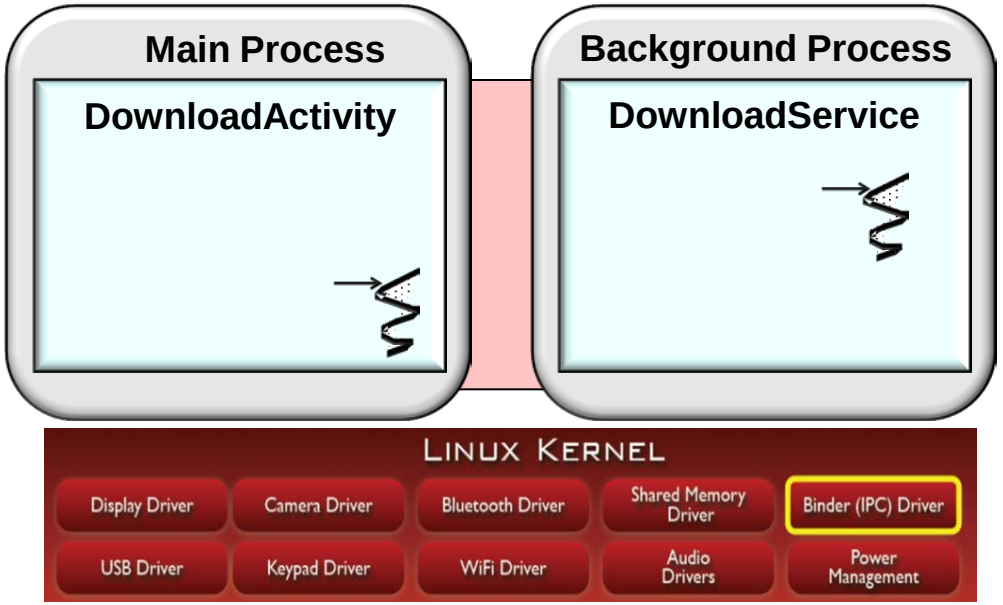
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients



Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
  - This choice is determined via a configuration setting in AndroidManifest.xml

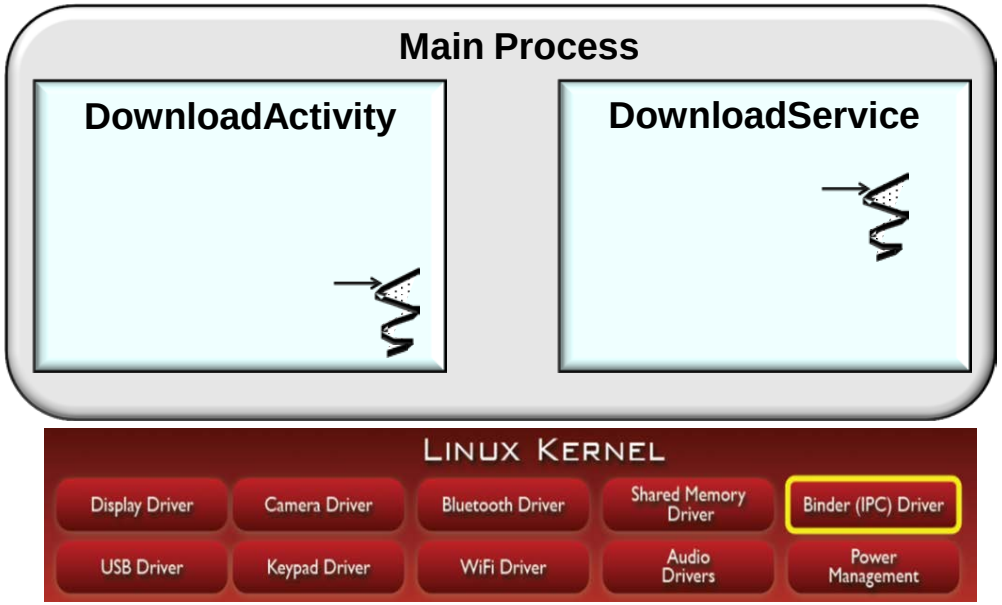



```
<service
    android:enabled
      =["true" | "false"]
    android:exported
      =["true" | "false"]
    android:icon="drawable resource"
    android:isolatedProcess=["true" | "false"]
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:process="string" >
    ...
</service>
```

developer.android.com/guide/topics/manifest/service-element.html has more

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
  - This choice is determined via a configuration setting in AndroidManifest.xml



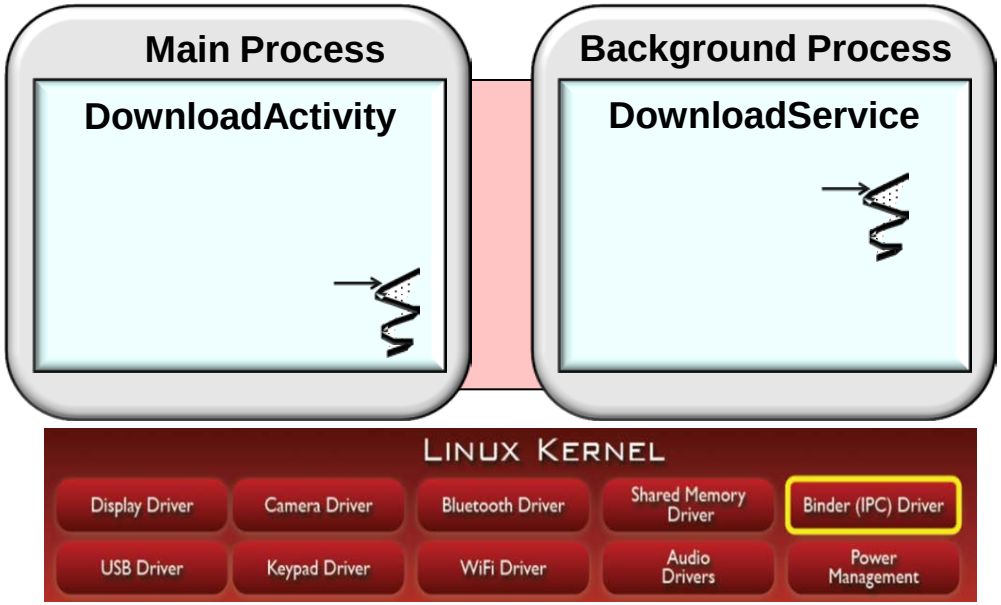
AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"

                        />
...
```

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
  - This choice is determined via a configuration setting in AndroidManifest.xml



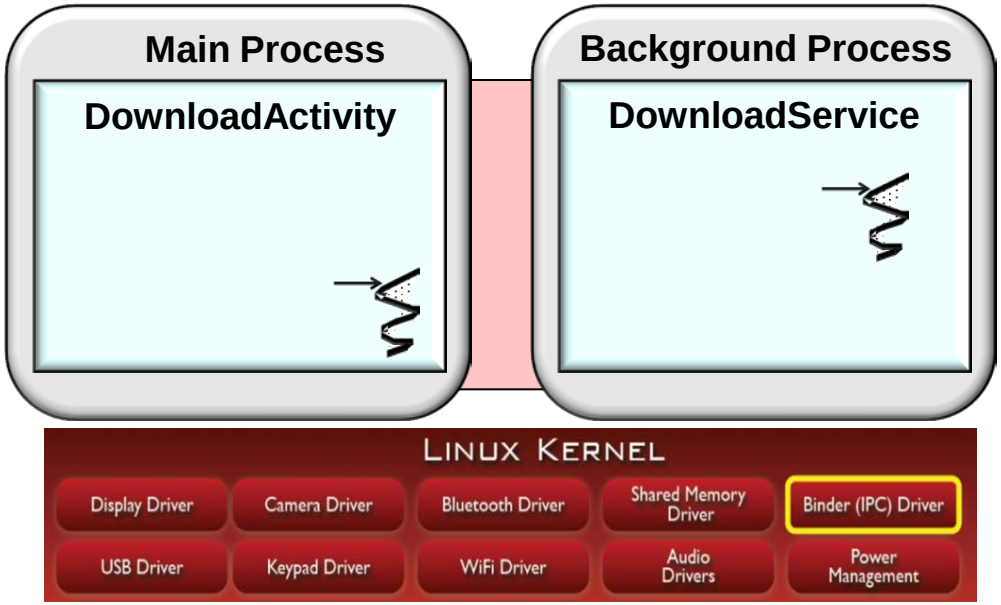
AndroidManifest.xml

```
...
<service android:name=
         "DownloadService"
         android:exported=
         "false"
         android:process=
         ":myProcess"/>
...
```

developer.android.com/guide/topics/manifest/service-element.html#proc

Overview of Service Deployment Models

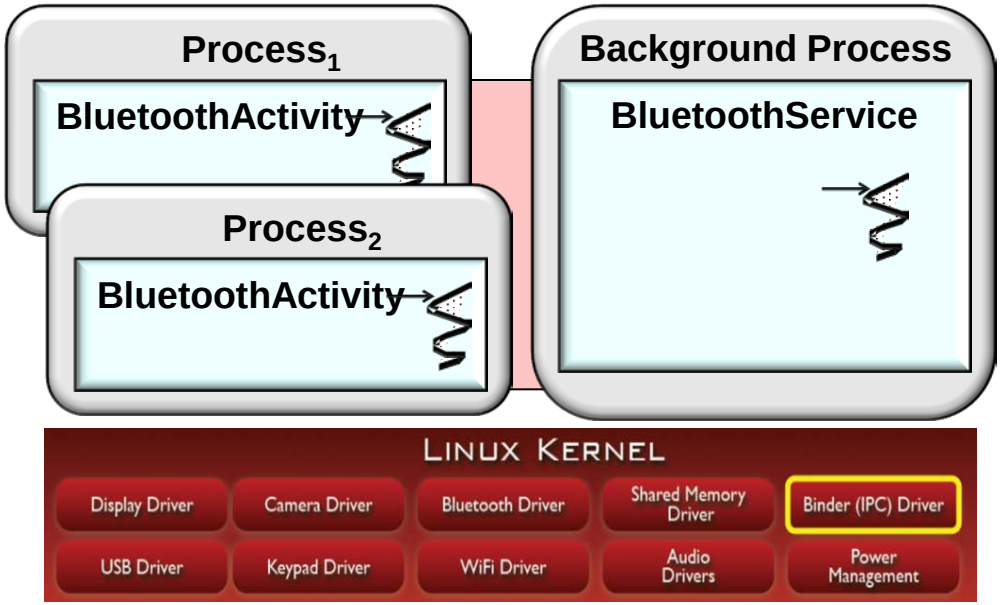
- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process



See www.vogella.com/tutorials/AndroidServices/article.html#service_advice

Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
  - Services shared by multiple applications need to run in separate processes

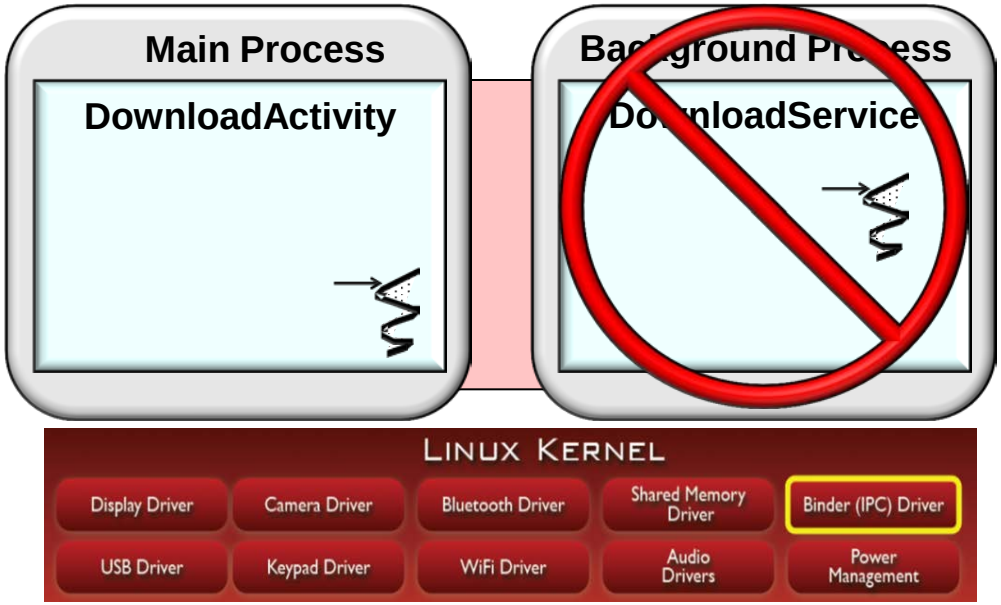



```
<service android:process="@string/process"
         android:name=".opp.BluetoothOppService"
         android:permission=
            "android.permission.ACCESS_BLUETOOTH_SHARE" />
```

See packages/apps/Bluetooth/AndroidManifest.xml

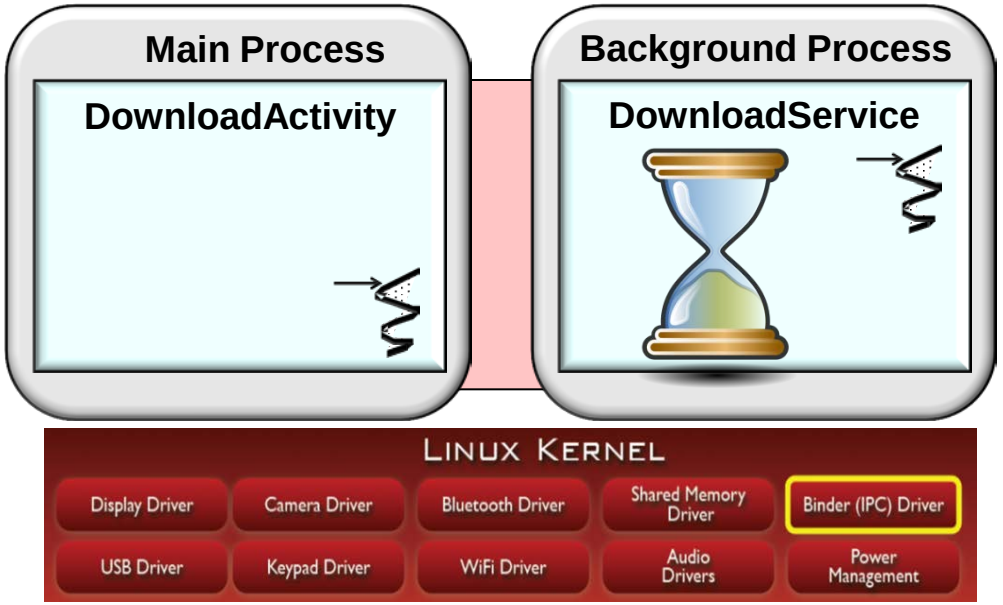
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
  - Services shared by multiple applications need to run in separate processes
  - Giving a Service its own address space can make applications more robust if failures or hangs occur



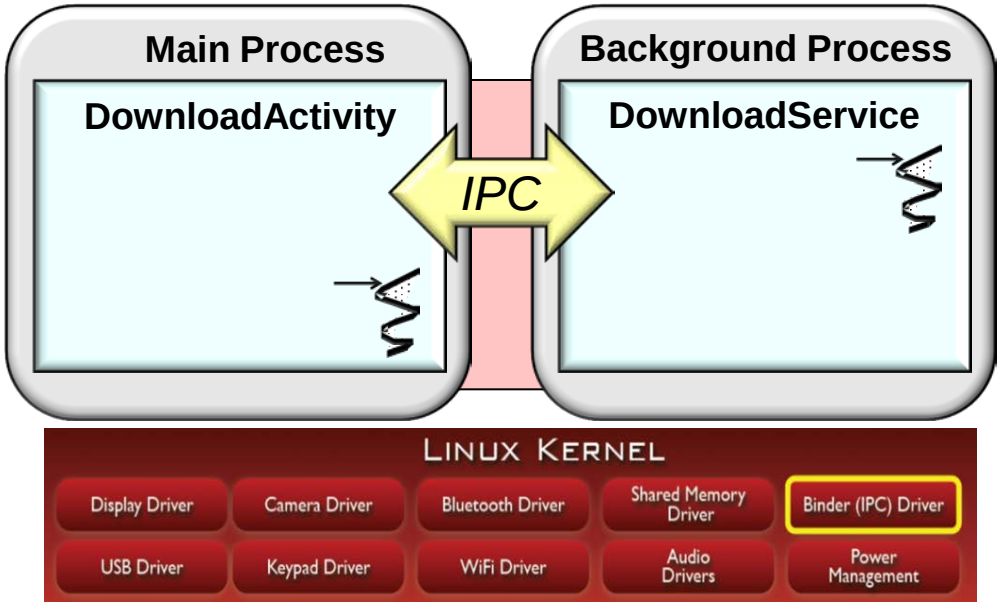
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
  - Services shared by multiple applications need to run in separate processes
  - Giving a Service its own address space can make applications more robust if failures or hangs occur
  - Garbage collection of the virtual machine in a separate Service process doesn't affect the Application process



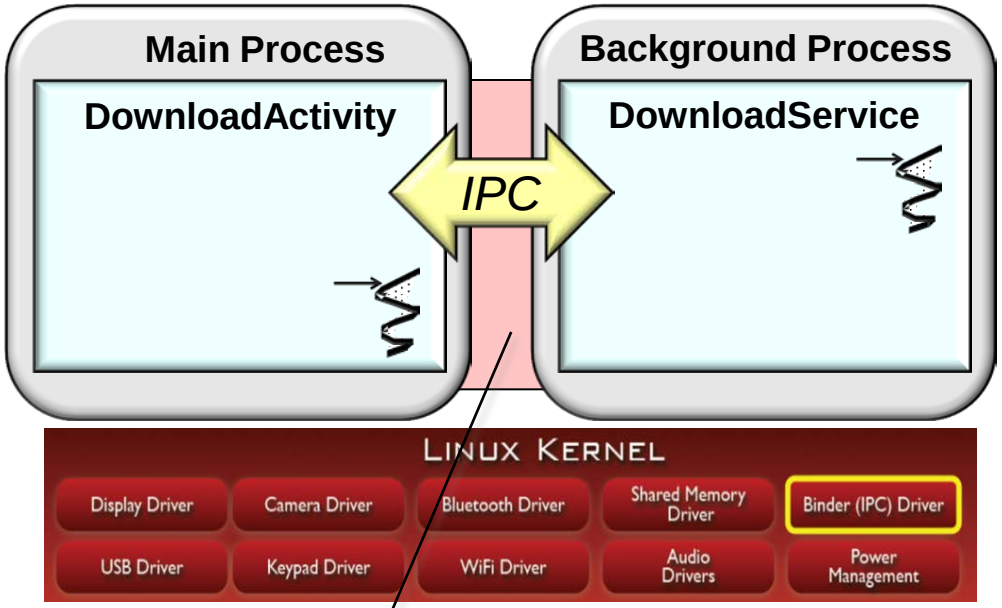
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes



Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - Android's Binder RPC framework underlies its various IPC mechanisms

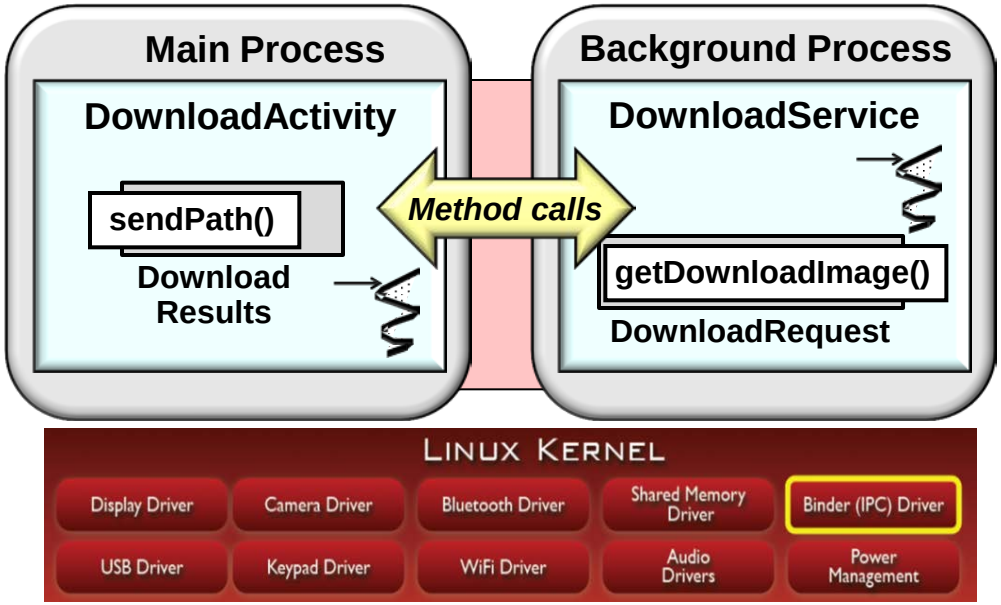


The Binder supports two-way or one-way client-service communication models

See elinux.org/Android_Binder for more on Binder

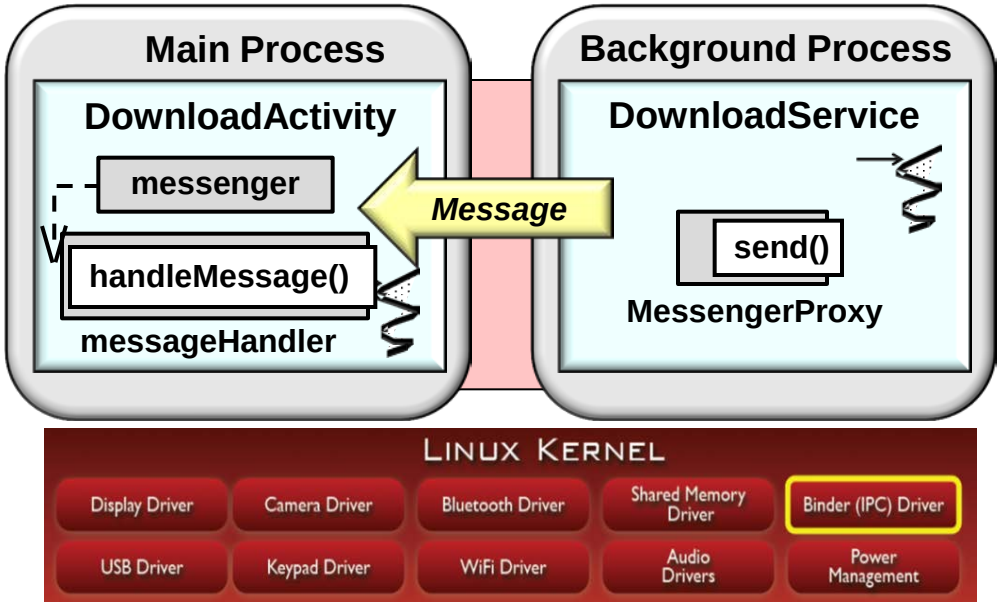
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - Android's Binder RPC framework underlies its various IPC mechanisms, e.g.
    - Synchronous & asynchronous remote method invocations via Android Interface Language Definition (AIDL)



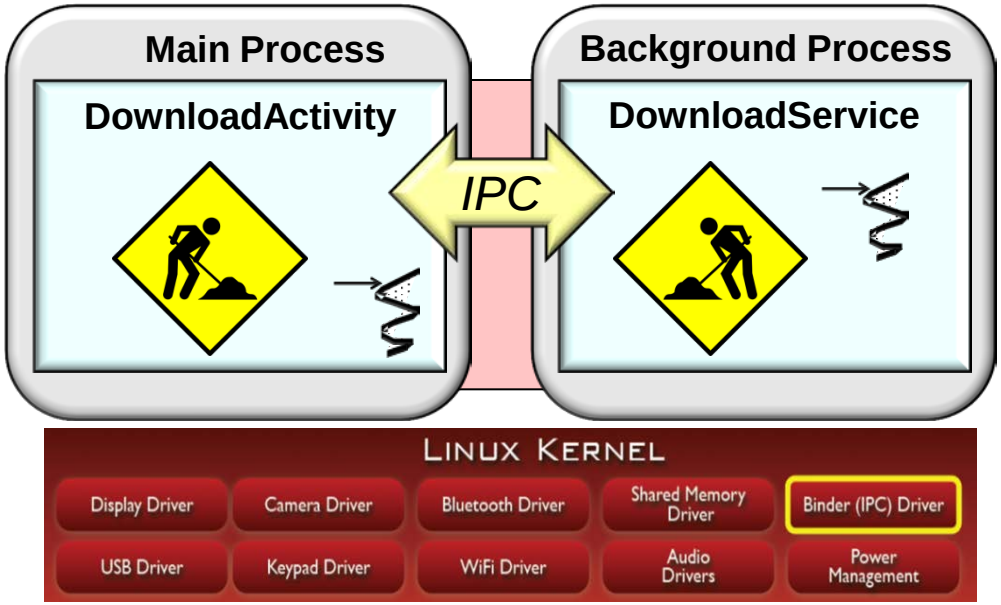
Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - Android's Binder RPC framework underlies its various IPC mechanisms, e.g.
    - Synchronous & asynchronous remote method invocations via Android Interface Language Definition (AIDL)
    - Asynchronous message passing via Android Messengers



Overview of Service Deployment Models

- Started & Bound Services can run in the same or different processes as their clients
- There are several reasons for running a Service in its own process
- IPC mechanisms are needed to communicate with Services running in different processes
  - Android's Binder RPC framework underlies its various IPC mechanisms

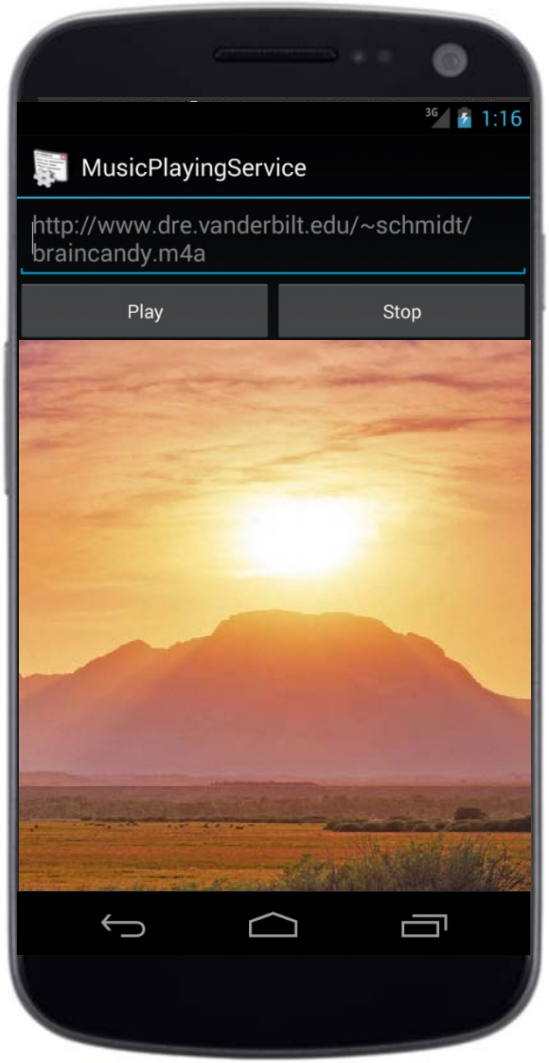


Running a Service in its own process may also require modifications to how data is exchanged

Analysis of the MusicPlayer Application (Part 1)

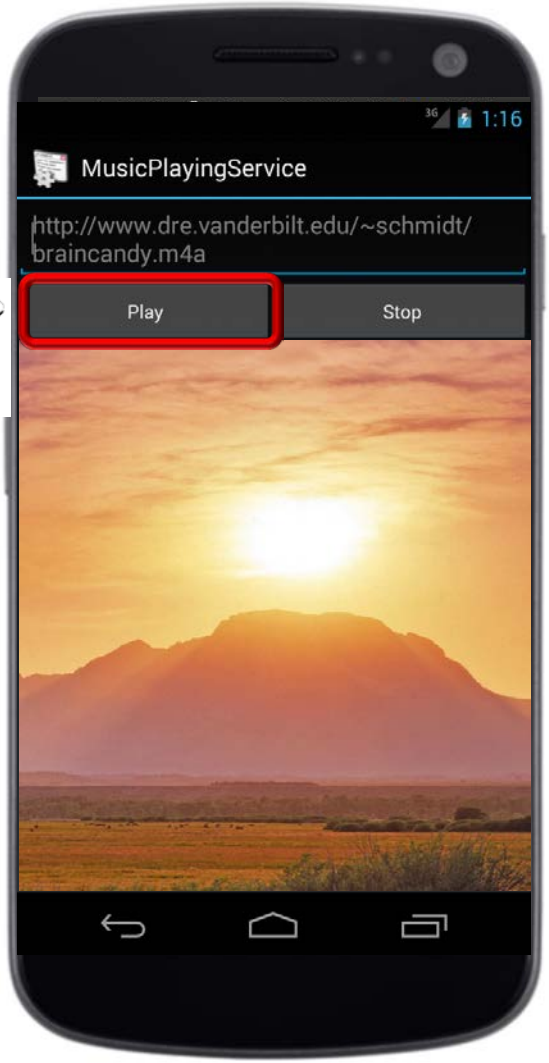
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service



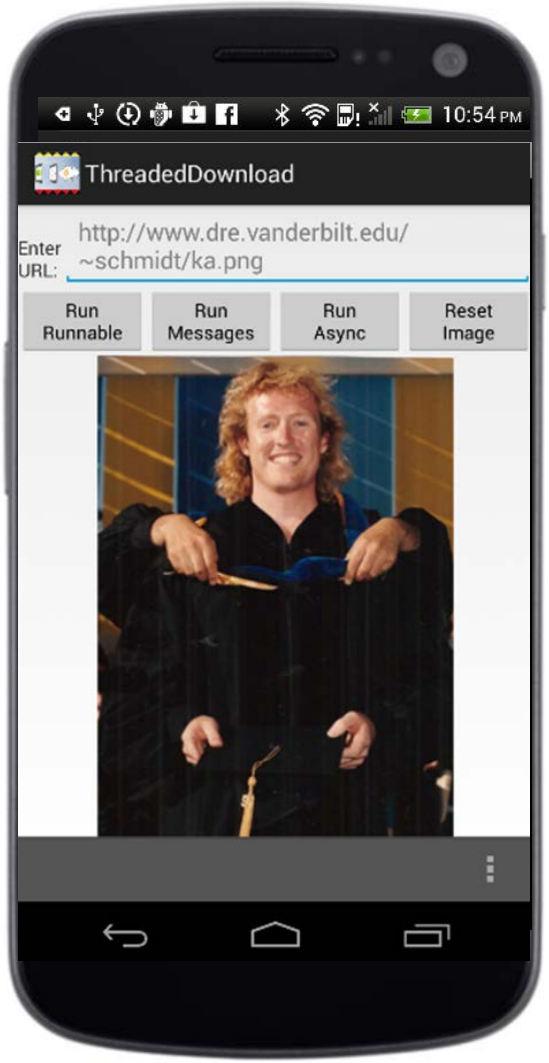
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
  - To start the Service a user needs to push the “Play” button



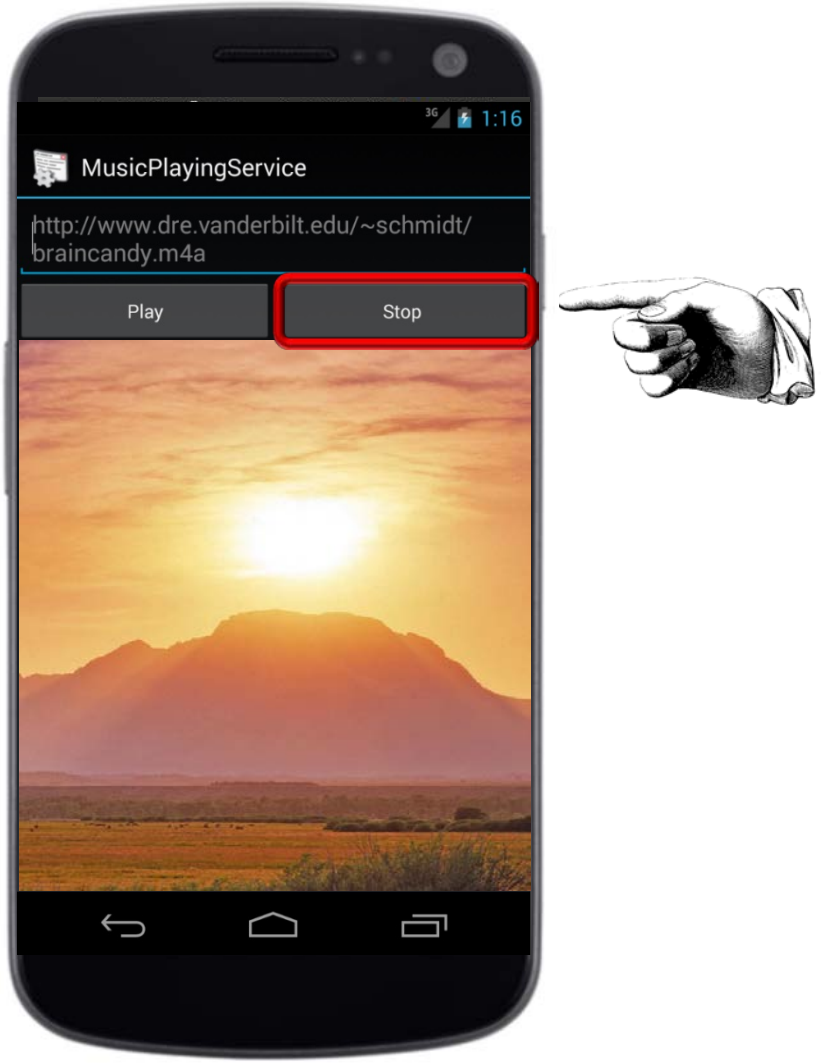
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
  - To start the Service a user needs to push the “Play” button
  - The Music Service will continue playing music when MusicActivity leaves the foreground



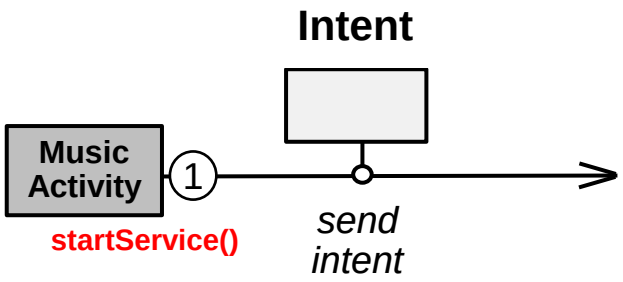
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
  - To start the Service a user needs to push the “Play” button
  - The Music Service will continue playing music when MusicActivity leaves the foreground
  - To stop the Service a user needs to explicitly push the “Stop” button



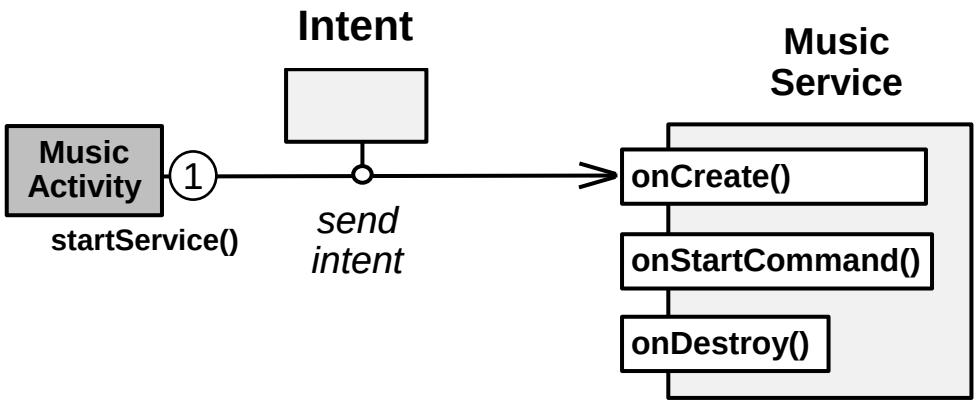
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
  - This Intent contains data that indicates which song to play



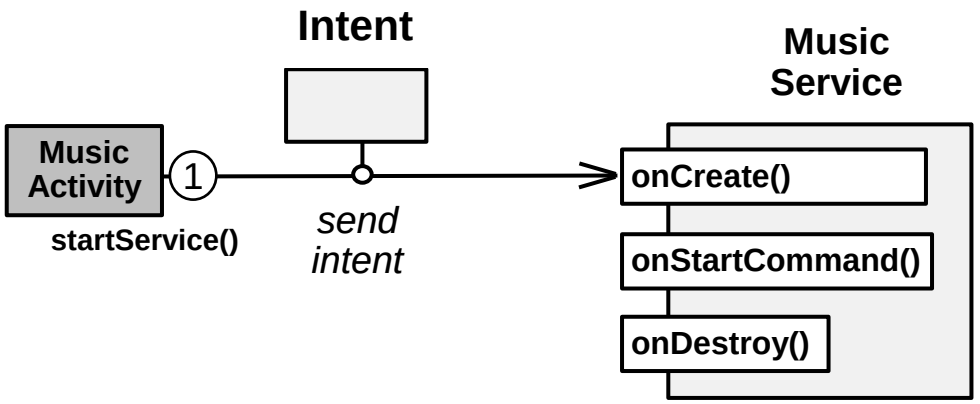
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand



Analysis of the Music Player Application

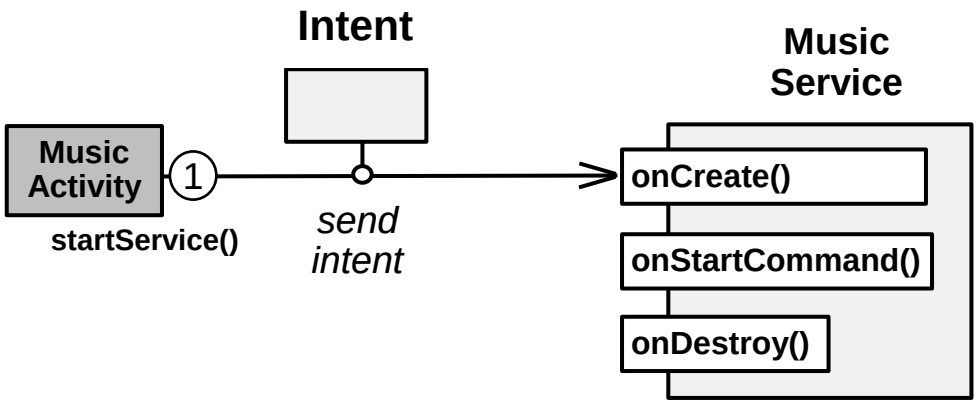
- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand
  - Based on the *Activator* pattern



See www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf

Analysis of the Music Player Application

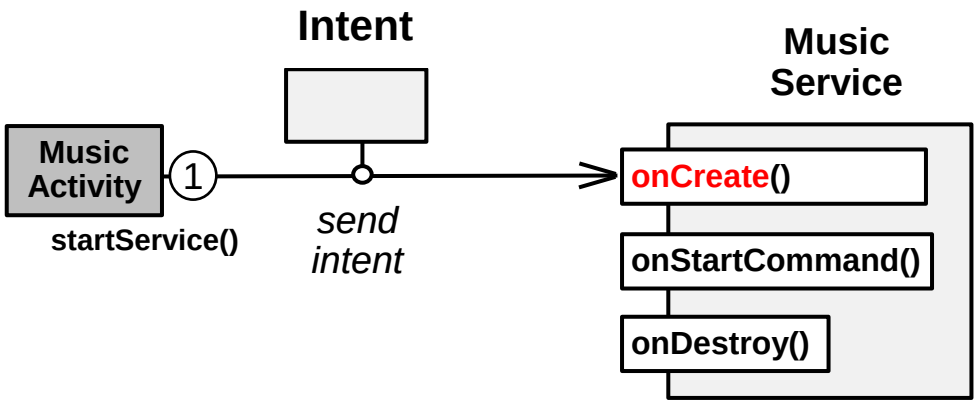
- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand
  - Based on the *Activator* pattern
    - Efficiently & transparently automates scalable on-demand activation & deactivation of services accessed by many clients



See www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf

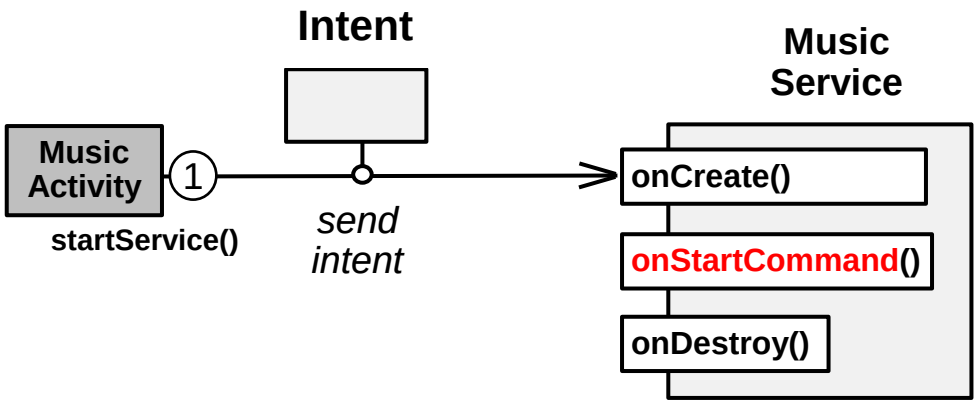
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand
  - Based on the *Activator* pattern
  - The onCreate() hook method is called when the MusicService is first launched



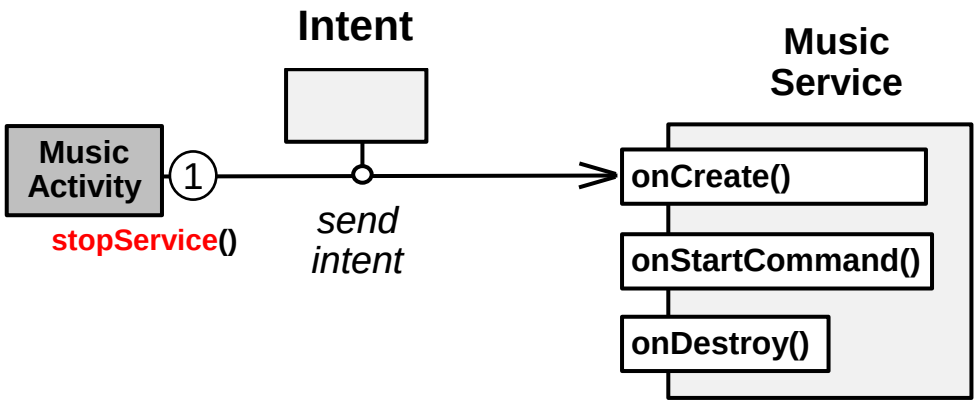
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand
  - Based on the *Activator* pattern
  - The onCreate() hook method is called when the MusicService is first launched
  - The onStartCommand() hook method initiates playing the song user requested via MusicActivity



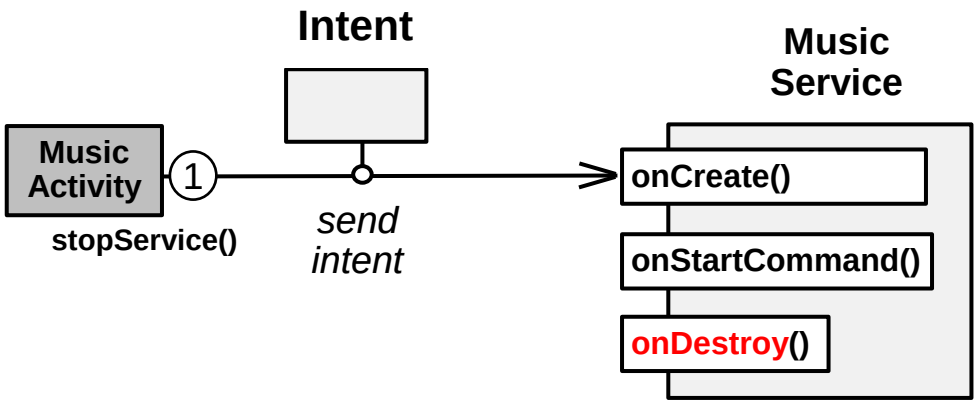
Analysis of the Music Player Application

- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand
- The MusicService is also stopped on-demand
  - Via a call to stopService()



Analysis of the Music Player Application

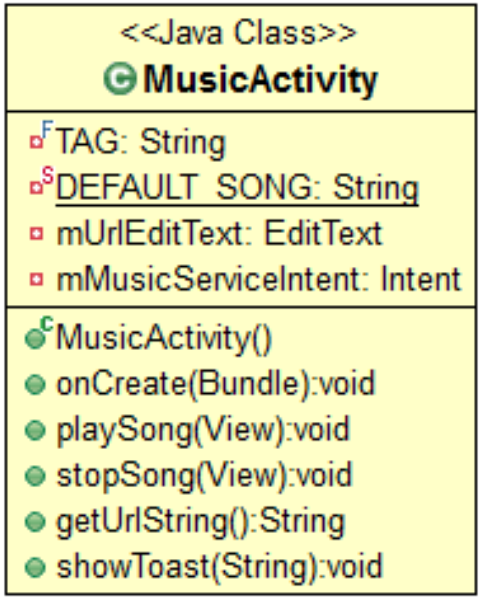
- MusicActivity can play music via a Started Service
- MusicActivity send an Intent via a call to startService()
- The MusicService is started on-demand
- The MusicService is also stopped on-demand
  - Via a call to stopService()
  - The onDestroy() hook method is called back to stop playing music

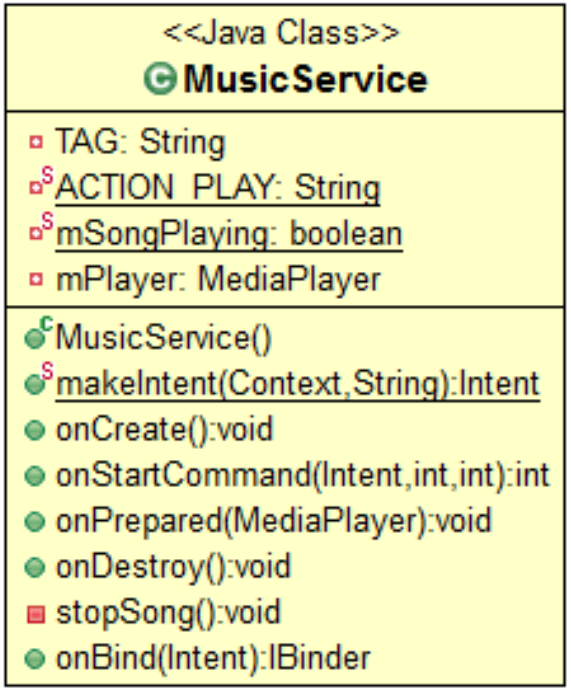


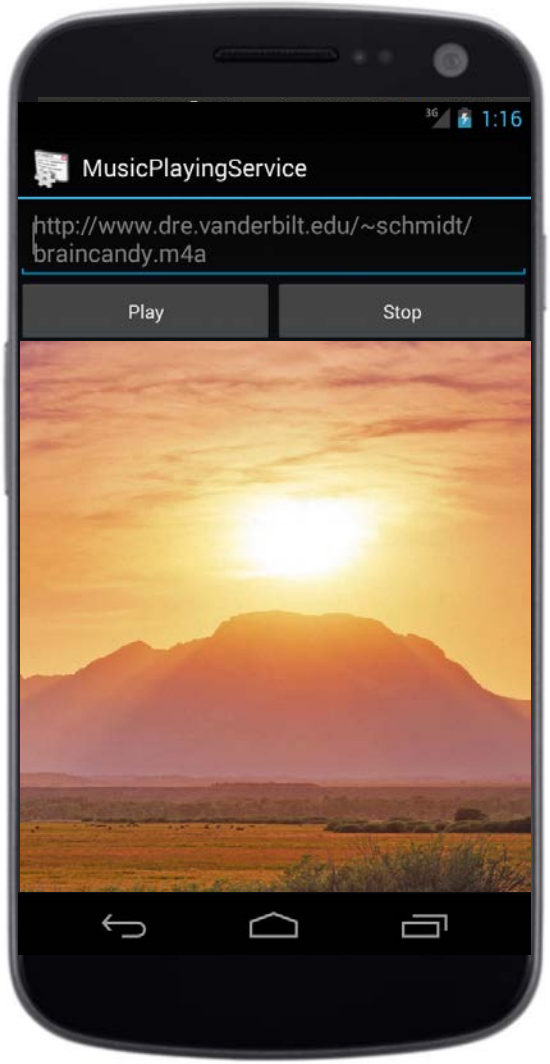
Analysis of the MusicPlayer Application (Part 2)

Analysis of the Music Player Application

- The MusicPlayer application contains a MusicActivity & a MusicService



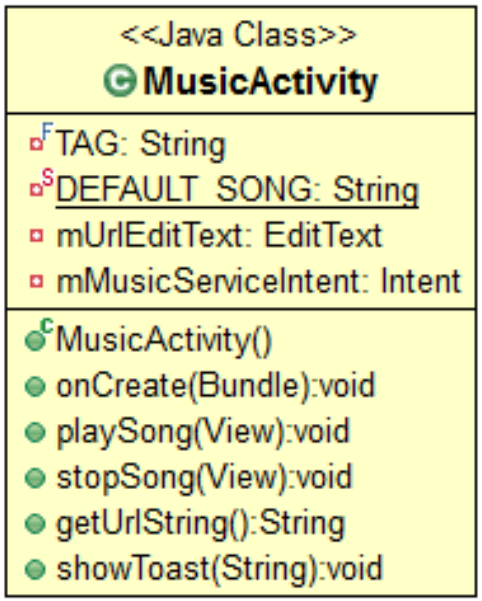


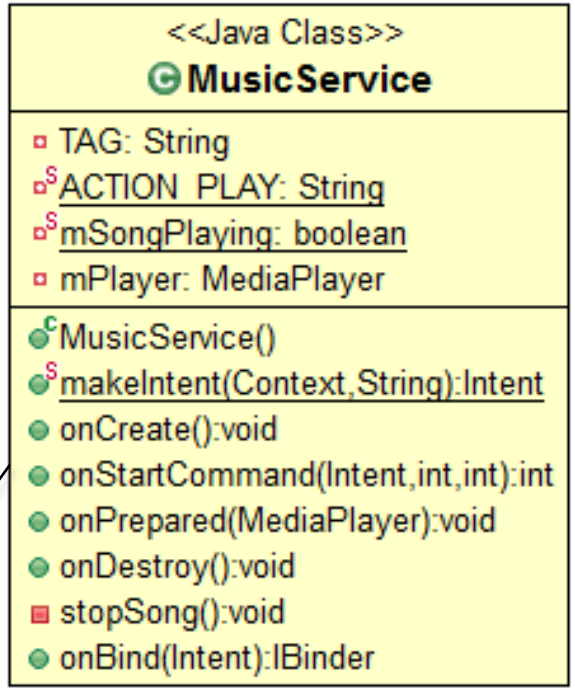


See github.com/douglascraigschmidt/CS251/tree/master/ex/MusicPlayingService

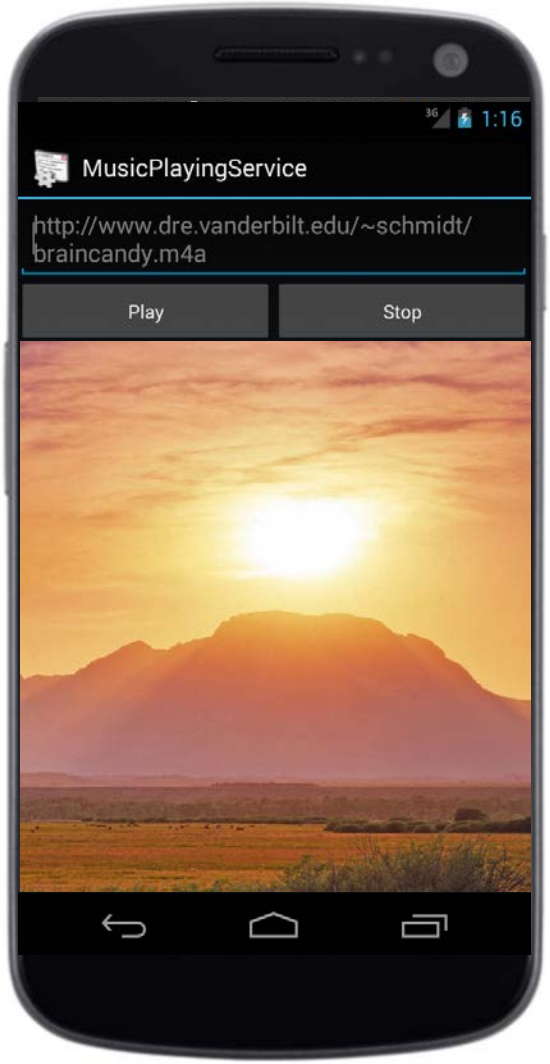
Analysis of the Music Player Application

- The MusicPlayer application contains a MusicActivity & a MusicService



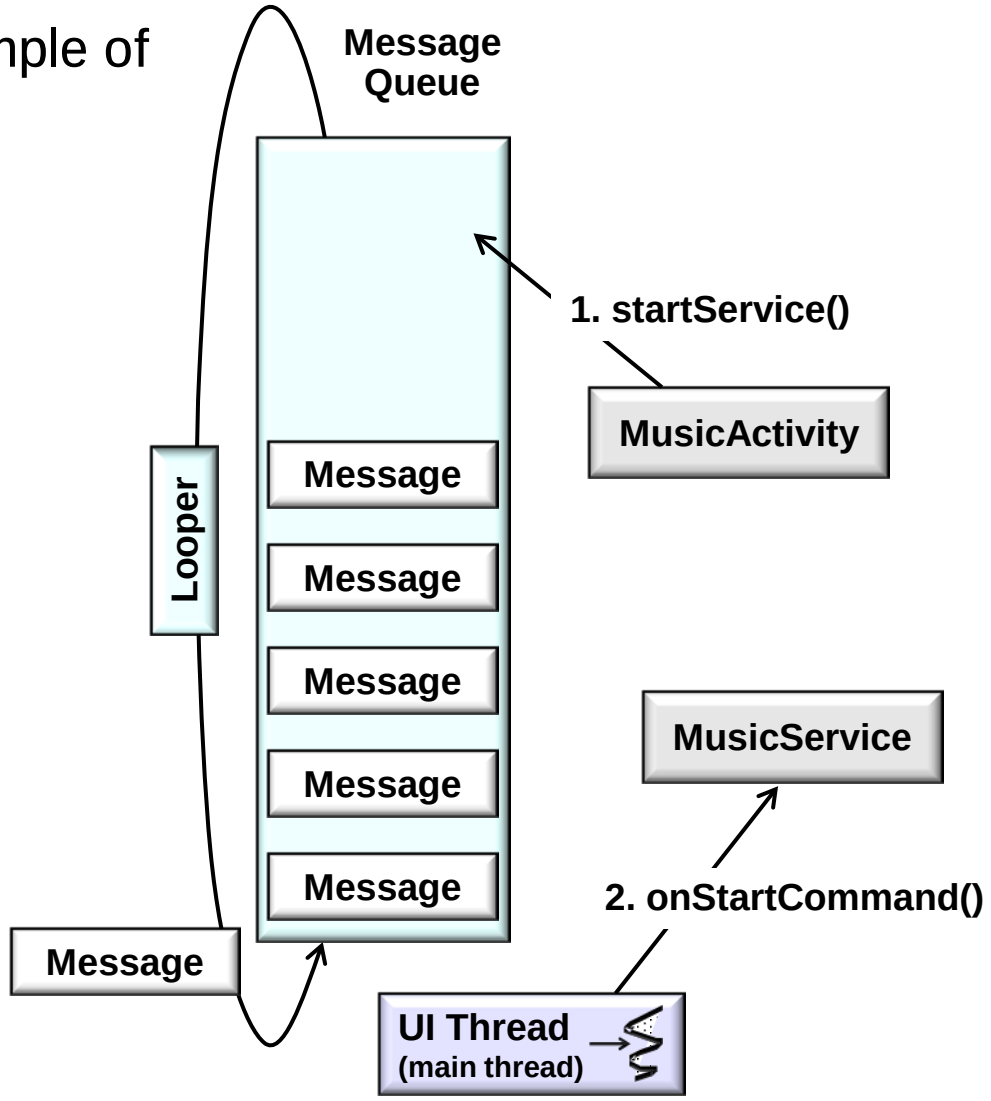


MusicService extends Service & implements MediaPlayer.OnPreparedListener to avoid blocking the UI Thread during initial streaming



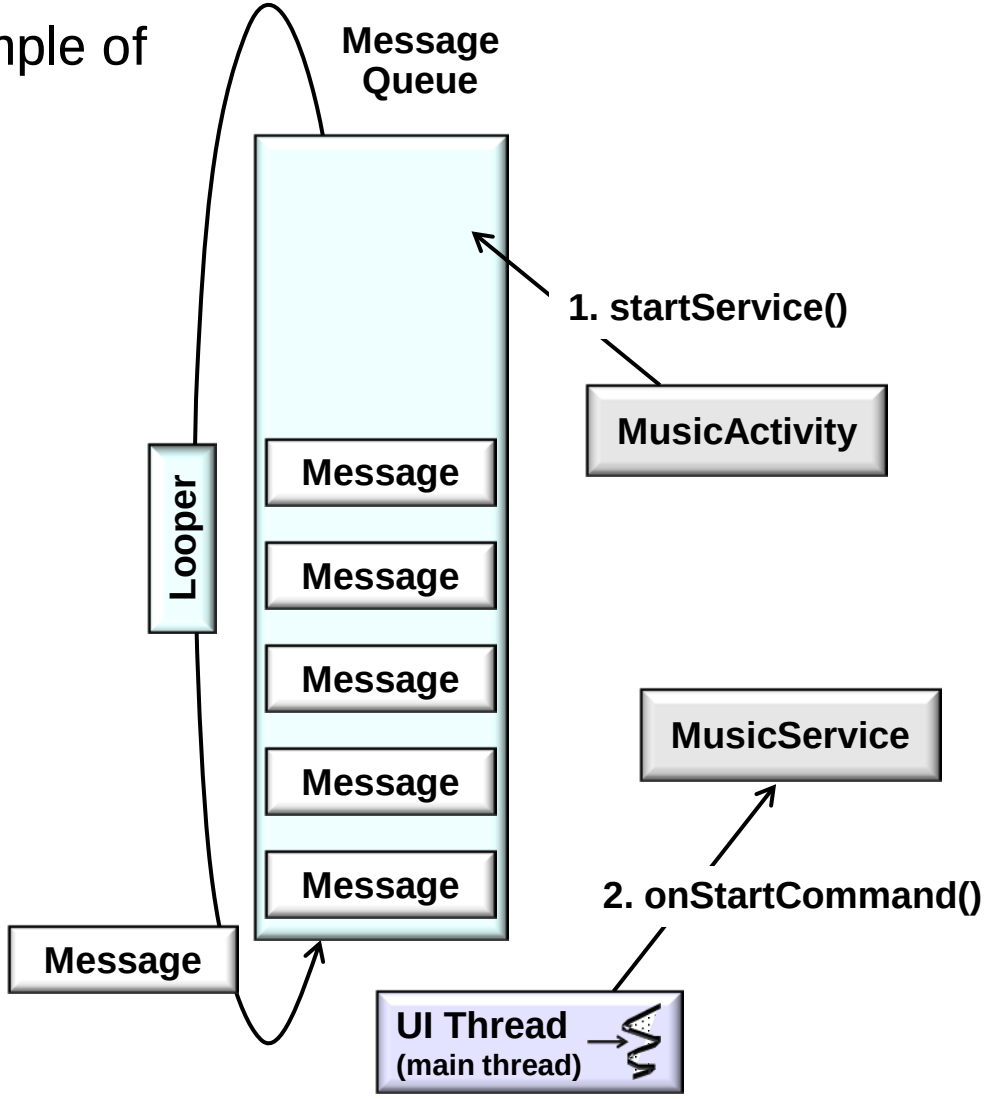
Analysis of the Music Player Application

- The MusicPlayer application contains a MusicActivity & a MusicService
- This is a relatively simple example of a Started Service



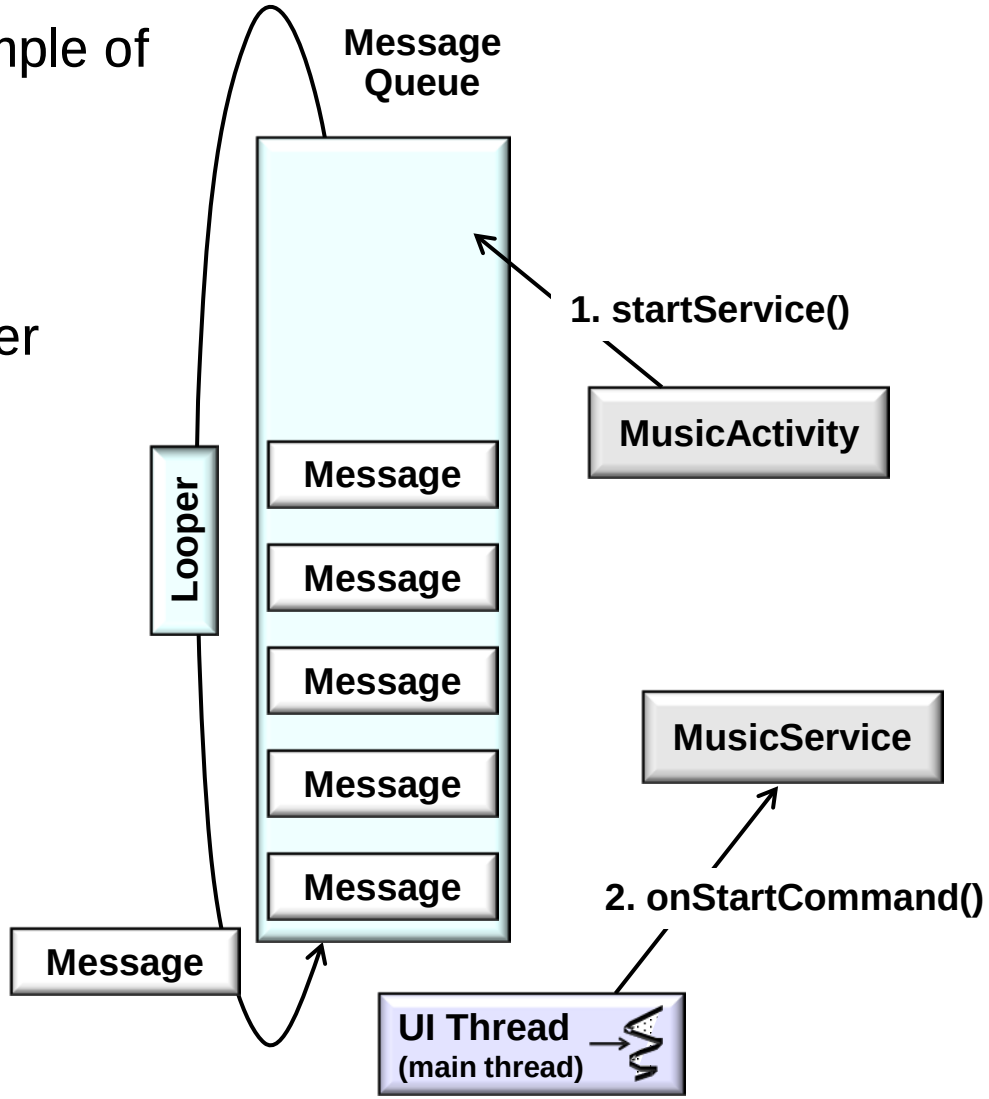
Analysis of the Music Player Application

- The MusicPlayer application contains a MusicActivity & a MusicService
- This is a relatively simple example of a Started Service, e.g.,
  - It doesn't need to spawn an internal threads explicitly



Analysis of the Music Player Application

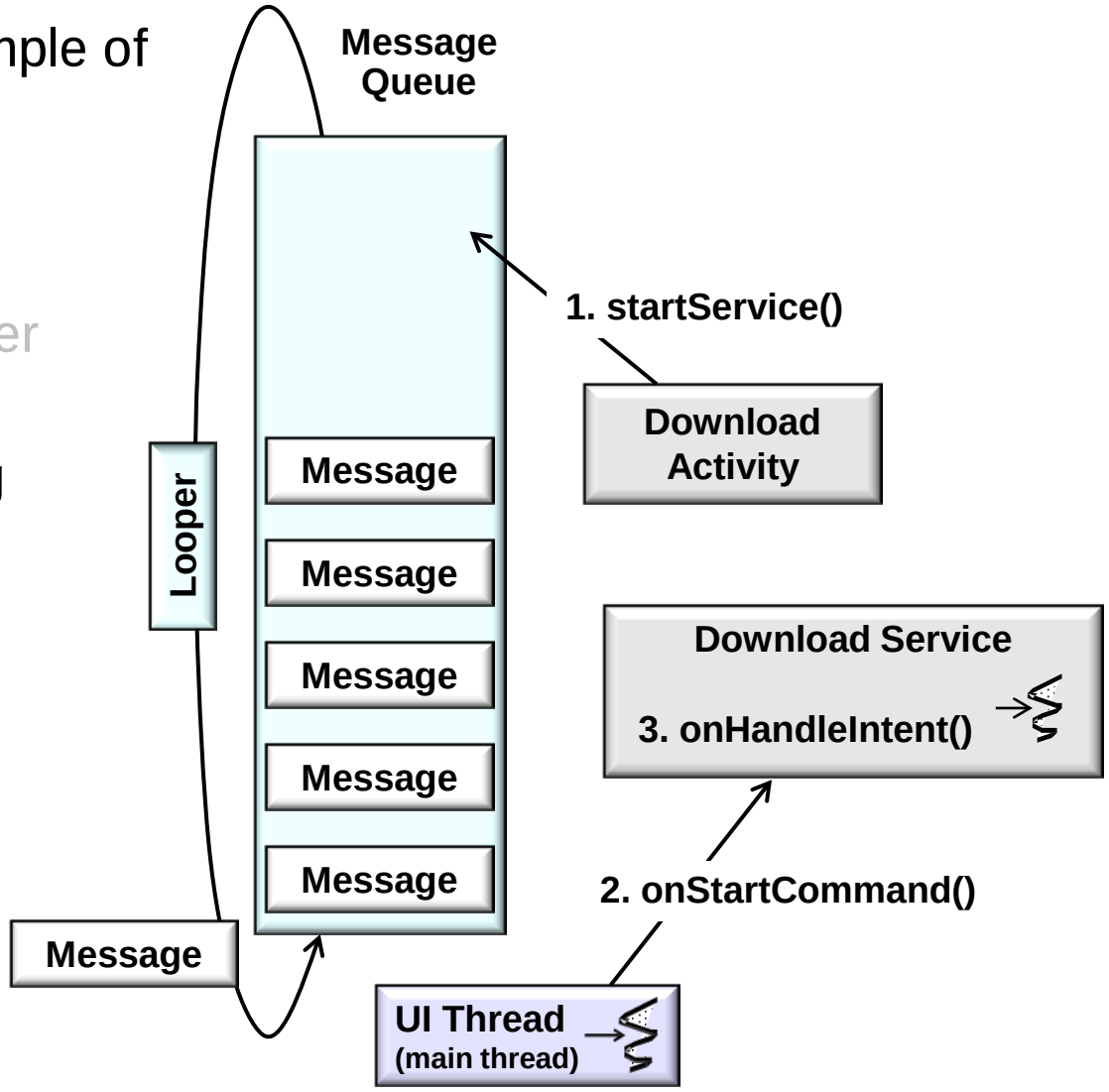
- The MusicPlayer application contains a MusicActivity & a MusicService
- This is a relatively simple example of a Started Service, e.g.,
  - It doesn't need to spawn an internal threads explicitly
    - Instead, it uses MediaPlayer asynchrony features



See developer.android.com/guide/topics/media/mediaplayer.html#asyncprepare

Analysis of the Music Player Application

- The MusicPlayer application contains a MusicActivity & a MusicService
- This is a relatively simple example of a Started Service, e.g.,
  - It doesn't need to spawn an internal threads explicitly
    - Instead, it uses MediaPlayer asynchrony features
  - Services with long-running operations may need to run in separate thread(s)



See developer.android.com/guide/components/services.html#ExtendingIntentService

Analysis of the Music Player Application

- The MusicPlayer application contains a MusicActivity & a MusicService
- This is a relatively simple example of a Started Service, e.g.,
  - It doesn't need to spawn an internal threads explicitly
  - There's no communication from the Service back to the Activity that invoked it