# Services

Service是一个可以在后台进行长时间呢操作的应用程序组件，无用户界面。可以被另一个应用组件启动，即使切换到另一个app依然可以在后台运行。另外，一个组件可绑定到service上并与之交互甚至可执行进程见通信（IPC）,例如，service可在后台播放音乐，下载文件，执行io操作等

A service can essentially take two forms:

Started

应用程序组件可以调用startService()开启一个服务。一旦开启服务，就不可控了。即使开启服务的组件销毁。通常来说，一个被start的服务只执行一个操作，且不向开启他的组件返回任何结果。一旦操作结束，service应该自行停止。

Bound

调用bindService()也可以开启服务，一个被bound开启的服务提供了一个用户-服务器接口，以允许调和调用它的组件进行互动。发送请求，就收结果甚至夸进程通信。一个被bound的service一旦开启它的组件被销毁，该服务也被销毁。多个徐建可绑定到一个服务上来。，但是如果所有的都unbind了，服务也被销毁。

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()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) to allow components to start it and [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) to allow binding.

Regardless（不管） 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](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html). However, you can declare the service as private, in the manifest file, and block access from other applications. This is discussed more in the section about [Declaring the service in the manifest](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\services.html#Declaring).

**Caution:** （服务运行在主线程中）A service runs in the main thread of its hosting process—the service does **not** create its own thread and does **not** run in a separate process (unless you specify otherwise). This means that, if your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should create a new thread within the service to do that work. By using a separate thread, you will reduce the risk of Application Not Responding (ANR) errors and the application's main thread can remain dedicated(专用的) to user interaction with your activities.

## The Basics

To create a service, you must create a subclass of [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html)(or one of its existing subclasses). In your implementation, you need to override some callback methods that handle key aspects of the service lifecycle and provide a mechanism for components to bind to the service, if appropriate. The most important callback methods you should override are:

[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int))

The system calls this method when another component, such as an activity, requests that the service be started, by calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)). Once this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is done, by calling [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)). (If you only want to provide binding, you don't need to implement this method.)

[onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent))

The system calls this method when another component wants to bind with the service (such as to perform RPC), by calling [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). In your implementation of this method, you must provide an interface that clients use to communicate with the service, by returning an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html). You must always implement this method, but if you don't want to allow binding, then you should return null.

[onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onCreate())

The system calls this method when the service is first created, to perform one-time setup procedures (before it calls either [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) or [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent))). If the service is already running, this method is not called.

[onDestroy()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onDestroy())

The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc. This is the last call the service receives.

If a component starts the service by calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) (which results in a call to[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int))), then the service remains running until it stops itself with [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or another component stops it by calling [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)).

If a component calls [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)) to create the service (and [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) is not called), then the service runs only as long as the component is bound to it. Once the service is unbound from all clients, the system destroys it.

The Android system will force-stop a service only when memory is low and it must recover system resources for the activity that has user focus. If the service is bound to an activity that has user focus, then it's less likely to be killed, and if the service is declared to [run in the foreground](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\services.html#Foreground)(discussed later), then it will almost never be killed. Otherwise, if the service was started and is long-running, then the system will lower its position in the list of background tasks over time and the service will become highly susceptible to killing—if your service is started, then you must design it to gracefully handle restarts by the system. If the system kills your service, it restarts it as soon as resources become available again (though this also depends on the value you return from [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)), as discussed later). For more information about when the system might destroy a service, see the [Processes and Threading](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\processes-and-threads.html) document.

In the following sections, you'll see how you can create each type of service and how to use it from other application components.

### Declaring a service in the manifest

Like activities (and other components), you must declare all services in your application's manifest file.

To declare your service, add a [<service>](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\manifest\service-element.html) element as a child of the [<application>](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\manifest\application-element.html) element. For example:

<manifest ... >  
  ...  
  <application ... >  
      <service android:name=".ExampleService" />  
      ...  
  </application>  
</manifest>

There are other attributes you can include in the [<service>](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\manifest\service-element.html) element to define properties such as permissions required to start the service and the process in which the service should run. The[android:name](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\manifest\service-element.html#nm) attribute is the only required attribute—it specifies the class name of the service. Once you publish your application, you should not change this name, because if you do, you might break some functionality where explicit intents are used to reference your service (read the blog post, [Things That Cannot Change](http://android-developers.blogspot.com/2011/06/things-that-cannot-change.html)).

See the [<service>](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\manifest\service-element.html) element reference（参考） for more information about declaring your service in the manifest.

Just like an activity, a service can define intent filters that allow other components to invoke the service using implicit intents. By declaring intent filters, components from any application installed on the user's device can potentially start your service if your service declares an intent filter that matches the intent another application passes to [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)).

If you plan on using your service only locally (other applications do not use it), then you don't need to (and should not) supply any intent filters. Without any intent filters, you must start the service using an intent that explicitly names the service class. More information about [starting a service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\services.html#StartingAService) is discussed below.

Additionally, you can ensure that your service is private to your application only if you include the [android:exported](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\guide\\topics\\manifest\\service-element.html" \l "exported) attribute and set it to "false". This is effective even if your service supplies intent filters.

For more information about creating intent filters for your service, see the [Intents and Intent Filters](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\intents-filters.html) document.

## Creating a Started Service

A started service is one that another component starts by calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)), resulting in a call to the service's [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) method.

When a service is started, it has a lifecycle that's independent of the component that started it and the service can run in the background indefinitely, even if the component that started it is destroyed. As such, the service should stop itself when its job is done by calling [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()), or another component can stop it by calling [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)).

An application component such as an activity can start the service by calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) and passing an [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html) that specifies the service and includes any data for the service to use. The service receives this [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html) in the [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) method.

For instance, suppose an activity needs to save some data to an online database. The activity can start a companion service and deliver it the data to save by passing an intent to [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)). The service receives the intent in [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)), connects to the Internet and performs the database transaction. When the transaction is done, the service stops itself and it is destroyed.

**Caution:** A services runs in the same process as the application in which it is declared and in the main thread of that application, by default. So, if your service performs intensive or blocking operations while the user interacts with an activity from the same application, the service will slow down activity performance. To avoid impacting application performance, you should start a new thread inside the service.

Traditionally, there are two classes you can extend to create a started service:

[Service](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html)

This is the base class for all services. When you extend this class, it's important that you create a new thread in which to do all the service's work, because the service uses your application's main thread, by default, which could slow the performance of any activity your application is running.

[IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html)

This is a subclass of [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html) that uses a worker thread to handle all start requests, one at a time. This is the best option if you don't require that your service handle multiple requests simultaneously. All you need to do is implement [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent)), which receives the intent for each start request so you can do the background work.

The following sections describe how you can implement your service using either one for these classes.

### Extending the IntentService class

Because most started services don't need to handle multiple requests simultaneously（同时的） (which can actually be a dangerous multi-threading scenario（方案）), it's probably best if you implement your service using the [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html) class.

The [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html) does the following:

* Creates a default worker thread that executes all intents delivered to [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) separate from your application's main thread.
* Creates a work queue that passes one intent at a time to your [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent)) implementation, so you never have to worry about multi-threading.
* Stops the service after all start requests have been handled, so you never have to call[stopSelf()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#stopSelf()).
* Provides default implementation of [onBind()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\IntentService.html#onBind(android.content.Intent)) that returns null.
* Provides a default implementation of [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onStartCommand(android.content.Intent, int, int)) that sends the intent to the work queue and then to your [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent)) implementation.

All this adds up to the fact that all you need to do is implement [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent)) to do the work provided by the client. (Though, you also need to provide a small constructor for the service.)

Here's an example implementation of [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html):

public class HelloIntentService extends IntentService {  
  
  /\*\*   
   \*   \* constructor with a name for the worker thread.  
   \*/  
  public HelloIntentService() {  
      super("HelloIntentService");  
  }  
  
  /\*\*  
   \* The IntentService calls this method from the default worker thread with  
   \* the intent that started the service. When this method returns, IntentService  
   \* stops the service, as appropriate.  
   \*/  
  @Override  
  protected void onHandleIntent(Intent intent) {  
      // Normally we would do some work here, like download a file.  
      // For our sample, we just sleep for 5 seconds.  
      long endTime = System.currentTimeMillis() + 5\*1000;  
      while (System.currentTimeMillis() < endTime) {  
          synchronized (this) {  
              try {  
                  wait(endTime - System.currentTimeMillis());  
              } catch (Exception e) {  
              }  
          }  
      }  
  }  
}

That's all you need: a constructor and an implementation of [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent)).

If you decide to also override other callback methods, such as [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onCreate()), [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onStartCommand(android.content.Intent, int, int)), or[onDestroy()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\IntentService.html#onDestroy()), be sure to call the super implementation, so that the [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html) can properly handle the life of the worker thread.

For example, [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onStartCommand(android.content.Intent, int, int)) must return the default implementation (which is how the intent gets delivered to [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent))):

@Override  
public int onStartCommand(Intent intent, int flags, int startId) {  
    Toast.makeText(this, "service starting", Toast.LENGTH\_SHORT).show();  
    return super.onStartCommand(intent,flags,startId);  
}

Besides [onHandleIntent()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html" \l "onHandleIntent(android.content.Intent)), the only method from which you don't need to call the super class is[onBind()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\IntentService.html#onBind(android.content.Intent)) (but you only need to implement that if your service allows binding).

In the next section, you'll see how the same kind of service is implemented when extending the base[Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html) class, which is a lot more code, but which might be appropriate if you need to handle simultaneous start requests.

### Extending the Service class

As you saw in the previous section, using [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html) makes your implementation of a started service very simple. If, however, you require your service to perform multi-threading (instead of processing start requests through a work queue), then you can extend the [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html) class to handle each intent.

For comparison, the following example code is an implementation of the [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html) class that performs the exact same work as the example above using [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html). That is, for each start request, it uses a worker thread to perform the job and processes only one request at a time.

public class HelloService extends Service {  
  private Looper mServiceLooper;  
  private ServiceHandler mServiceHandler;  
  
  // Handler that receives messages from the thread  
  private final class ServiceHandler extends Handler {  
      public ServiceHandler(Looper looper) {  
          super(looper);  
      }  
      @Override  
      public void handleMessage(Message msg) {  
          // Normally we would do some work here, like download a file.  
          // For our sample, we just sleep for 5 seconds.  
          long endTime = System.currentTimeMillis() + 5\*1000;  
          while (System.currentTimeMillis() < endTime) {  
              synchronized (this) {  
                  try {  
                      wait(endTime - System.currentTimeMillis());  
                  } catch (Exception e) {  
                  }  
              }  
          }  
          // Stop the service using the startId, so that we don't stop  
          // the service in the middle of handling another job  
          stopSelf(msg.arg1);  
      }  
  }  
  
  @Override  
  public void onCreate() {  
    // Start up the thread running the service.  Note that we create a  
    // separate thread because the service normally runs in the process's  
    // main thread, which we don't want to block.  We also make it  
    // background priority so CPU-intensive work will not disrupt our UI.  
    HandlerThread thread = new HandlerThread("ServiceStartArguments",  
            Process.THREAD\_PRIORITY\_BACKGROUND);  
    thread.start();  
      
    // Get the HandlerThread's Looper and use it for our Handler   
    mServiceLooper = thread.getLooper();  
    mServiceHandler = new ServiceHandler(mServiceLooper);  
  }  
  
  @Override  
  public int onStartCommand(Intent intent, int flags, int startId) {  
      Toast.makeText(this, "service starting", Toast.LENGTH\_SHORT).show();  
  
      // For each start request, send a message to start a job and deliver the  
      // start ID so we know which request we're stopping when we finish the job  
      Message msg = mServiceHandler.obtainMessage();  
      msg.arg1 = startId;  
      mServiceHandler.sendMessage(msg);  
        
      // If we get killed, after returning from here, restart  
      return START\_STICKY;  
  }  
  
  @Override  
  public IBinder onBind(Intent intent) {  
      // We don't provide binding, so return null  
      return null;  
  }  
    
  @Override  
  public void onDestroy() {  
    Toast.makeText(this, "service done", Toast.LENGTH\_SHORT).show();   
  }  
}

As you can see, it's a lot more work than using [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html).

However, because you handle each call to [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) yourself, you can perform multiple requests simultaneously. That's not what this example does, but if that's what you want, then you can create a new thread for each request and run them right away (instead of waiting for the previous request to finish).

Notice that the [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) method must return an integer. The integer is a value that describes how the system should continue the service in the event that the system kills it (as discussed above, the default implementation for [IntentService](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\IntentService.html) handles this for you, though you are able to modify it). The return value from [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) must be one of the following constants:

[START\_NOT\_STICKY](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "START_NOT_STICKY)

If the system kills the service after [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) returns, do not recreate the service, unless there are pending（悬而未决） 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](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "START_STICKY)

If the system kills the service after [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) returns, recreate the service and call[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int)), but do not redeliver the last intent. Instead, the system calls [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int))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](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "START_REDELIVER_INTENT)

If the system kills the service after [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) returns, recreate the service and call[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int)) 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.

For more details about these return values, see the linked reference documentation for each constant.

### Starting a Service

You can start a service from an activity or other application component by passing an [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html)(specifying the service to start) to [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)). The Android system calls the service's[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int)) method and passes it the [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html). (You should never call [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) directly.)

For example, an activity can start the example service in the previous section (HelloSevice) using an explicit intent with [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)):

Intent intent = new Intent(this, HelloService.class);  
startService(intent);

The [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) method returns immediately and the Android system calls the service's[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int)) method. If the service is not already running, the system first calls [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onCreate()), then calls [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)).

If the service does not also provide binding, the intent delivered with [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) is the only mode of communication between the application component and the service. However, if you want the service to send a result back, then the client that starts the service can create a [PendingIntent](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\PendingIntent.html)for a broadcast (with [getBroadcast()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\PendingIntent.html" \l "getBroadcast(android.content.Context, int, android.content.Intent, int))) and deliver it to the service in the [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html) that starts the service. The service can then use the broadcast to deliver a result.

Multiple requests to start the service result in multiple corresponding calls to the service's[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int)). **However, only one request to stop the service (with [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent))) is required to stop it.**

### Stopping a service

A started service must manage its own lifecycle. That is, the system does not stop or destroy the service unless it must recover system memory and the service continues to run after [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int))returns. So, the service must stop itself by calling [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or another component can stop it by calling [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)).

Once requested to stop with [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)), the system destroys the service as soon as possible.

**However, if your service handles multiple requests to [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) concurrently, then you shouldn't stop the service when you're done processing a start request, because you might have since received a new start request (stopping at the end of the first request would terminate the second one). To avoid this problem, you can use [stopSelf(int)](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf(int)) to ensure that your request to stop the service is always based on the most recent start request. That is, when you call [stopSelf(int)](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf(int)), you pass the ID of the start request (the startId delivered to [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int))) to which your stop request corresponds. Then if the service received a new start request before you were able to call**[**stopSelf(int)**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#stopSelf(int))**, then the ID will not match and the service will not stop.**

**Caution:** It's important that your application stops its services when it's done working, to avoid wasting system resources and consuming battery power. If necessary, other components can stop the service by calling [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)). Even if you enable binding for the service, you must always stop the service yourself if it ever received a call to [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)).

For more information about the lifecycle of a service, see the section below about [Managing the Lifecycle of a Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\services.html#Lifecycle).

## Creating a Bound Service

A bound service is one that allows application components to bind to it by calling [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)) in order to create a long-standing connection (and generally does not allow components to start it by calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent))).

You should create a bound service when you want to interact with the service from activities and other components in your application or to expose some of your application's functionality to other applications, through interprocess communication (IPC).

To create a bound service, you must implement the [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) callback method to return an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) that defines the interface for communication with the service. Other application components can then call[bindService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) to retrieve the interface and begin calling methods on the service. The service lives only to serve the application component that is bound to it, so when there are no components bound to the service, the system destroys it (you do not need to stop a bound service in the way you must when the service is started through [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int))).

To create a bound service, the first thing you must do is define the interface that specifies how a client can communicate with the service. This interface between the service and a client must be an implementation of [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) and is what your service must return from the [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) callback method. Once the client receives the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html), it can begin interacting with the service through that interface.

Multiple clients can bind to the service at once. When a client is done interacting with the service, it calls [unbindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "unbindService(android.content.ServiceConnection)) to unbind. Once there are no clients bound to the service, the system destroys the service.

There are multiple ways to implement a bound service and the implementation is more complicated than a started service, so the bound service discussion appears in a separate document about [Bound Services](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\bound-services.html).

## Sending Notifications to the User

Once running, a service can notify the user of events using **[Toast Notifications](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\guide\\topics\\ui\\notifiers\\toasts.html) or**[**Status Bar Notifications**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\ui\notifiers\notifications.html)**.**

A toast notification is a message that appears on the surface of the current window for a moment then disappears, while a status bar notification provides an icon in the status bar with a message, which the user can select in order to take an action (such as start an activity).

Usually, a status bar notification is the best technique when some background work has completed (such as a file completed downloading) and the user can now act on it. When the user selects the notification from the expanded view, the notification can start an activity (such as to view the downloaded file).

See the [Toast Notifications](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\ui\notifiers\toasts.html) or [Status Bar Notifications](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\ui\notifiers\notifications.html) developer guides for more information.

## Running a Service in the Foreground

A foreground service is a service that's considered to be something the user is actively aware of and thus not a candidate（候补） for the system to kill when low on memory. A foreground service must provide a notification for the status bar, which is placed under the "Ongoing" heading, which means that the notification cannot be dismissed unless the service is either stopped or removed from the foreground.

For example, a music player that plays music from a service should be set to run in the foreground, because the user is explicitly aware of its operation. The notification in the status bar might indicate the current song and allow the user to launch an activity to interact with the music player.

To request that your service run in the foreground, call [startForeground()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "startForeground(int, android.app.Notification)). This method takes two parameters: an integer that uniquely identifies the notification and the [Notification](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Notification.html) for the status bar. For example:

Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker\_text),  
        System.currentTimeMillis());  
Intent notificationIntent = new Intent(this, ExampleActivity.class);  
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);  
notification.setLatestEventInfo(this, getText(R.string.notification\_title),  
        getText(R.string.notification\_message), pendingIntent);  
startForeground(ONGOING\_NOTIFICATION\_ID, notification);

**Caution:** The integer ID you give to [startForeground()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "startForeground(int, android.app.Notification)) must not be 0.

To remove the service from the foreground, call [stopForeground()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopForeground(boolean)). This method takes a boolean, indicating whether to remove the status bar notification as well. This method does not stop the service. However, if you stop the service while it's still running in the foreground, then the notification is also removed.

For more information about notifications, see [Creating Status Bar Notifications](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\topics\ui\notifiers\notifications.html).

## Managing the Lifecycle of a Service

The lifecycle of a service is much simpler than that of an activity. However, it's even more important that you pay close attention to how your service is created and destroyed, because a service can run in the background without the user being aware.

The service lifecycle—from when it's created to when it's destroyed—can follow two different paths:

* **A started service**

**The service is created when another component calls [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)). The service then runs indefinitely and must stop itself by calling [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()). Another component can also stop the service by calling [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)). When the service is stopped, the system destroys it.**

* **A bound service**

**The service is created when another component (a client) calls [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). The client then communicates with the service through an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) interface. The client can close the connection by calling [unbindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "unbindService(android.content.ServiceConnection)). Multiple clients can bind to the same service and when all of them unbind, the system destroys the service. (The service does not need to stop itself.)**

These two paths are not entirely separate. **That is, you can bind to a service that was already started with [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)).** For example, a background music service could be started by calling [startService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#startService(android.content.Intent)) with an [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html) that identifies the music to play. Later, possibly when the user wants to exercise some control over the player or get information about the current song, **an activity can bind to the service by calling [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). In cases like this, [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)) or [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) does not actually stop the service until all clients unbind.**

### Implementing the lifecycle callbacks

Like an activity, a service has lifecycle callback methods that you can implement to monitor changes in the service's state and perform work at the appropriate times. The following skeleton service demonstrates each of the lifecycle methods:

public class ExampleService extends Service {  
    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  
  
    @Override  
    public void [onCreate](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onCreate())() {  
        // The service is being created  
    }  
    @Override  
    public int [onStartCommand](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int))(Intent intent, int flags, int startId) {  
        **// The service is starting, due to a call to** [**startService()**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#startService(android.content.Intent)) **return *mStartMode*;**  
    }  
    @Override  
    public IBinder [onBind](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onBind(android.content.Intent))(Intent intent) {  
        // A client is binding to the service with [bindService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int))  
        return *mBinder*;  
    }  
    @Override  
    public boolean [onUnbind](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onUnbind(android.content.Intent))(Intent intent) {  
        // All clients have unbound with [unbindService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#unbindService(android.content.ServiceConnection))  
        return *mAllowRebind*;  
    }  
    @Override  
    public void [onRebind](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onRebind(android.content.Intent))(Intent intent) {  
        // A client is binding to the service with [bindService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)),  
        // after onUnbind() has already been called  
    }  
    @Override  
    public void [onDestroy](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onDestroy())() {  
        // The service is no longer used and is being destroyed  
    }  
}

**Note:** **Unlike the activity lifecycle callback methods, you are not required to call the superclass implementation of these callback methods.**



**Figure 2.** The service lifecycle. The diagram on the left shows the lifecycle when the service is created with [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) and the diagram on the right shows the lifecycle when the service is created with[bindService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)).

By implementing these methods, you can monitor two nested loops of the service's lifecycle:

* **The entire lifetime of a service happens between the time [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onCreate()) is called and the time**[**onDestroy()**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onDestroy())**returns. Like an activity, a service does its initial setup in [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onCreate()) and releases all remaining resources in [onDestroy()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onDestroy()). For example, a music playback service could create the thread where the music will be played in [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onCreate()), then stop the thread in [onDestroy()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onDestroy()).**

The [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onCreate()) and [onDestroy()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onDestroy()) methods are called for all services, whether they're created by[startService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#startService(android.content.Intent)) or [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)).

* **The active lifetime of a service begins with a call to either [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) or [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)). Each method is handed the**[**Intent**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html)**that was passed to either [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) or [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)), respectively（各自）.**

**If the service is started, the active lifetime ends the same time that the entire lifetime ends (the service is still active even after [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) returns). If the service is bound, the active lifetime ends when [onUnbind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onUnbind(android.content.Intent)) returns.**

**Note:** Although a started service is stopped by a call to either [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)), there is not a respective（各自的） callback for the service (there's no onStop() callback). So, unless the service is bound to a client, the system destroys it when the service is stopped—[onDestroy()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onDestroy()) is the only callback received.

Figure 2 illustrates the typical callback methods for a service. Although the figure separates services that are created by [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)) from those created by [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)), keep in mind that any service, no matter how it's started, can potentially allow clients to bind to it. So, a service that was initially started with [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) (by a client calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent))) can still receive a call to [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) (when a client calls [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int))).

For more information about creating a service that provides binding, see the [Bound Services](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\bound-services.html)document, which includes more information about the [onRebind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onRebind(android.content.Intent)) callback method in the section about[Managing the Lifecycle of a Bound Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\bound-services.html#Lifecycle).

# Bound Services

A bound service is an implementation of the [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html) class that allows other applications to bind to it and interact with it. **To provide binding for a service, you must implement the [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) callback method. This method returns an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) object that defines the programming interface that clients can use to interact with the service**.

### Binding to a Started Service

As discussed in the [Services](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\services.html) document, you can create a service that is both started and bound. That is, the service can be started by calling [startService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "startService(android.content.Intent)), which allows the service to run indefinitely, and also allow a client to bind to the service by calling [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)).

If you do allow your service to be started and bound, then when the service has been started, the system does not destroy the service when all clients unbind. Instead, you must explicitly stop the service, by calling [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf())or [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)).

Although you should usually implement either [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) or[onStartCommand()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onStartCommand(android.content.Intent, int, int)), it's sometimes necessary to implement both. For example, a music player might find it useful to allow its service to run indefinitely and also provide binding. This way, an activity can start the service to play some music and the music continues to play even if the user leaves the application. Then, when the user returns to the application, the activity can bind to the service to regain control of playback.

Be sure to read the section about[Managing the Lifecycle of a Bound Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\bound-services.html#Lifecycle), for more information about the service lifecycle when adding binding to a started service.

A client can bind to the service by calling [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). When it does, it must provide an implementation of[ServiceConnection](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ServiceConnection.html), which monitors the connection with the service. The [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)) method returns immediately without a value, but when the Android system creates the connection between the client and service, it calls [onServiceConnected()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html" \l "onServiceConnected(android.content.ComponentName, android.os.IBinder))on the [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html), to deliver the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) that the client can use to communicate with the service.

Multiple clients can connect to the service at once. However, the system calls your service's [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) method to retrieve the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) only when the first client binds. The system then delivers the same [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) to any additional clients that bind, without calling [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) again.

When the last client unbinds from the service, the system destroys the service (unless the service was also started by[startService()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#startService(android.content.Intent))).

When you implement your bound service, the most important part is defining the interface that your [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) callback method returns. There are a few different ways you can define your service's [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) interface and the following section discusses each technique.

## Creating a Bound Service

When creating a service that provides binding, you must provide an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) that provides the programming interface that clients can use to interact with the service. There are three ways you can define the interface:

[Extending the Binder class](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\guide\\components\\bound-services.html" \l "Binder)

If your service is private to your own application and runs in the same process as the client (which is common), you should create your interface by extending the [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) class and returning an instance of it from[onBind()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onBind(android.content.Intent)). The client receives the [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) and can use it to directly access public methods available in either the [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) implementation or even the [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html).

This is the preferred（首选的） technique when your service is merely a background worker for your own application. The only reason you would not create your interface this way is because your service is used by other applications or across separate processes.

[Using a Messenger](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\guide\\components\\bound-services.html" \l "Messenger)

If you need your interface to work across different processes, you can create an interface for the service with a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html). In this manner（习惯、风俗）, the service defines a[Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html) that responds to different types of [Message](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Message.html)objects. This [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html) is the basis for a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) that can then share an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) with the client, allowing the client to send commands to the service using [Message](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Message.html)objects. Additionally, the client can define a [Messenger](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Messenger.html)of its own so the service can send messages back.

This is the simplest way to perform interprocess communication (IPC), because the [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) queues all requests into a single thread so that you don't have to design your service to be thread-safe.

Using AIDL

AIDL (Android Interface Definition Language) performs all the work to decompose objects into primitives that the operating system can understand and marshall them across processes to perform IPC. The previous technique, **using a**[**Messenger**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html)**, is actually based on AIDL as its underlying structure**. As mentioned above, the [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) creates a queue of all the client requests in a single thread, so the service receives requests one at a time. If, however, you want your service to handle multiple requests simultaneously（同时的）, then you can use AIDL directly. In this case, your service must be capable of multi-threading and be built thread-safe.

To use AIDL directly, you must create an .aidl file that defines the programming interface. The Android SDK tools use this file to generate an abstract class that implements the interface and handles IPC, which you can then extend within your service.

**Note:** Most applications **should not** use AIDL to create a bound service, because it may require multithreading capabilities and can result in a more complicated implementation. As such, AIDL is not suitable for most applications and this document does not discuss how to use it for your service. If you're certain that you need to use AIDL directly, see the [AIDL](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\aidl.html) document.

### Extending the Binder class

If your service is used only by the local application and does not need to work across processes, then you can implement your own [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) class that provides your client direct access to public methods in the service.

**Note:** This works only if the client and service are in the same application and process, which is most common. For example, this would work well for a music application that needs to bind an activity to its own service that's playing music in the background.

Here's how to set it up:

1. In your service, create an instance of [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) that either:
   * contains public methods that the client can call
   * returns the current [Service](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html) instance, which has public methods the client can call
   * or, returns an instance of another class hosted by the service with public methods the client can call
2. Return this instance of [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) from the [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) callback method.
3. In the client, receive the [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) from the [onServiceConnected()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html" \l "onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback method and make calls to the bound service using the methods provided.

**Note:** The reason the service and client must be in the same application is so the client can cast the returned object and properly call its APIs. The service and client must also be in the same process, because this technique does not perform any marshalling across processes.

For example, here's a service that provides clients access to methods in the service through a [Binder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Binder.html) implementation:

public class LocalService extends Service {  
    // Binder given to clients  
    private final IBinder mBinder = new LocalBinder();  
    // Random number generator  
    private final Random mGenerator = new Random();  
  
    /\*\*  
     \* Class used for the client Binder.  Because we know this service always  
     \* runs in the same process as its clients, we don't need to deal with IPC.  
     \*/  
    public class LocalBinder extends Binder {  
        LocalService getService() {  
            // Return this instance of LocalService so clients can call public methods  
            return LocalService.this;  
        }  
    }  
  
    @Override  
    public IBinder onBind(Intent intent) {  
        return mBinder;  
    }  
  
    /\*\* method for clients \*/  
    public int getRandomNumber() {  
      return mGenerator.nextInt(100);  
    }  
}

The LocalBinder provides the getService() method for clients to retrieve the current instance of LocalService. This allows clients to call public methods in the service. For example, clients can call getRandomNumber() from the service.

Here's an activity that binds to LocalService and calls getRandomNumber() when a button is clicked:

public class BindingActivity extends Activity {  
    LocalService mService;  
    boolean mBound = false;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        // Bind to LocalService  
        Intent intent = new Intent(this, LocalService.class);  
        bindService(intent, mConnection, Context.BIND\_AUTO\_CREATE);  
    }  
  
    @Override  
    protected void onStop() {  
        super.onStop();  
        // Unbind from the service  
        if (mBound) {  
            unbindService(mConnection);  
            mBound = false;  
        }  
    }  
  
    /\*\* Called when a button is clicked (the button in the layout file attaches to  
      \* this method with the android:onClick attribute) \*/  
    public void onButtonClick(View v) {  
        if (mBound) {  
            // Call a method from the LocalService.  
            // However, if this call were something that might hang, then this request should  
            // occur in a separate thread to avoid slowing down the activity performance.  
            int num = mService.getRandomNumber();  
            Toast.makeText(this, "number: " + num, Toast.LENGTH\_SHORT).show();  
        }  
    }  
  
    /\*\* Defines callbacks for service binding, passed to bindService() \*/  
    private ServiceConnection mConnection = new ServiceConnection() {  
  
        @Override  
        public void onServiceConnected(ComponentName className,  
                IBinder service) {  
            // We've bound to LocalService, cast the IBinder and get LocalService instance  
            LocalBinder binder = (LocalBinder) service;  
            mService = binder.getService();  
            mBound = true;  
        }  
  
        @Override  
        public void onServiceDisconnected(ComponentName arg0) {  
            mBound = false;  
        }  
    };  
}

The above sample shows how the client binds to the service using an implementation of[ServiceConnection](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ServiceConnection.html) and the [onServiceConnected()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html" \l "onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback. The next section provides more information about this process of binding to the service.

**Note:** The example above doesn't explicitly unbind from the service, but all clients should unbind at an appropriate time (such as when the activity pauses).

For more sample code, see the [LocalService.java](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\LocalService.html) class and the [LocalServiceActivities.java](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\LocalServiceActivities.html) class in[ApiDemos](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\index.html).

### Using a Messenger

#### Compared to AIDL

When you need to perform IPC, using a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) for your interface is simpler than implementing it with AIDL, because[Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) queues all calls to the service, whereas, a pure AIDL interface sends simultaneous requests to the service, which must then handle multi-threading.

For most applications, the service doesn't need to perform multi-threading, so using a [Messenger](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Messenger.html)allows the service to handle one call at a time. If it's important that your service be multi-threaded, then you should use [AIDL](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\guide\\components\\aidl.html)to define your interface.

If you need your service to communicate with remote processes, then you can use a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) to provide the interface for your service. This technique allows you to perform interprocess communication (IPC) without the need to use AIDL.

Here's a summary of how to use a [Messenger](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Messenger.html):

* The service implements a [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html) that receives a callback for each call from a client.
* The [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html) is used to create a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) object (which is a reference to the [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html)).
* The [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) creates an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) that the service returns to clients from [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)).
* Clients use the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) to instantiate the [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) (that references the service's [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html)), which the client uses to send [Message](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Message.html) objects to the service.
* The service receives each [Message](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Message.html) in its [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html)—specifically, in the [handleMessage()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Handler.html" \l "handleMessage(android.os.Message)) method.

In this way, there are no "methods" for the client to call on the service. Instead, the client delivers "messages" ([Message](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Message.html) objects) that the service receives in its [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html).

Here's a simple example service that uses a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) interface:

public class MessengerService extends Service {  
    /\*\* Command to the service to display a message \*/  
    static final int MSG\_SAY\_HELLO = 1;  
  
    /\*\*  
     \* Handler of incoming messages from clients.  
     \*/  
    class IncomingHandler extends Handler {  
        @Override  
        public void handleMessage(Message msg) {  
            switch (msg.what) {  
                case MSG\_SAY\_HELLO:  
                    Toast.makeText(getApplicationContext(), "hello!", Toast.LENGTH\_SHORT).show();  
                    break;  
                default:  
                    super.handleMessage(msg);  
            }  
        }  
    }  
  
    /\*\*  
     \* Target we publish for clients to send messages to IncomingHandler.  
     \*/  
    final Messenger mMessenger = new Messenger(new IncomingHandler());  
  
    /\*\*  
     \* When binding to the service, we return an interface to our messenger  
     \* for sending messages to the service.  
     \*/  
    @Override  
    public IBinder onBind(Intent intent) {  
        Toast.makeText(getApplicationContext(), "binding", Toast.LENGTH\_SHORT).show();  
        return mMessenger.getBinder();  
    }  
}

Notice that the [handleMessage()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Handler.html" \l "handleMessage(android.os.Message)) method in the [Handler](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Handler.html) is where the service receives the incoming[Message](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Message.html) and decides what to do, based on the [what](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Message.html#what) member.

All that a client needs to do is create a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) based on the [IBinder](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\IBinder.html) returned by the service and send a message using [send()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html#send(android.os.Message)). For example, here's a simple activity that binds to the service and delivers the MSG\_SAY\_HELLO message to the service:

public class ActivityMessenger extends Activity {  
    /\*\* Messenger for communicating with the service. \*/  
    Messenger mService = null;  
  
    /\*\* Flag indicating whether we have called bind on the service. \*/  
    boolean mBound;  
  
    /\*\*  
     \* Class for interacting with the main interface of the service.  
     \*/  
    private ServiceConnection mConnection = new ServiceConnection() {  
        public void onServiceConnected(ComponentName className, IBinder service) {  
            // This is called when the connection with the service has been  
            // established, giving us the object we can use to  
            // interact with the service.  We are communicating with the  
            // service using a Messenger, so here we get a client-side  
            // representation of that from the raw IBinder object.  
            mService = new Messenger(service);  
            mBound = true;  
        }  
  
        public void onServiceDisconnected(ComponentName className) {  
            // This is called when the connection with the service has been  
            // unexpectedly disconnected -- that is, its process crashed.  
            mService = null;  
            mBound = false;  
        }  
    };  
  
    public void sayHello(View v) {  
        if (!mBound) return;  
        // Create and send a message to the service, using a supported 'what' value  
        Message msg = Message.obtain(null, MessengerService.MSG\_SAY\_HELLO, 0, 0);  
        try {  
            mService.send(msg);  
        } catch (RemoteException e) {  
            e.printStackTrace();  
        }  
    }  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
  
    @Override  
    protected void onStart() {  
        super.onStart();  
        // Bind to the service  
        bindService(new Intent(this, MessengerService.class), mConnection,  
            Context.BIND\_AUTO\_CREATE);  
    }  
  
    @Override  
    protected void onStop() {  
        super.onStop();  
        // Unbind from the service  
        if (mBound) {  
            unbindService(mConnection);  
            mBound = false;  
        }  
    }  
}

Notice that this example does not show how the service can respond to the client. If you want the service to respond, then you need to also create a [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) in the client. Then when the client receives the [onServiceConnected()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html" \l "onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback, it sends a [Message](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Message.html) to the service that includes the client's [Messenger](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html) in the [replyTo](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\Message.html" \l "replyTo) parameter of the [send()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\os\Messenger.html#send(android.os.Message)) method.

You can see an example of how to provide two-way messaging in the [MessengerService.java](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\MessengerService.html) (service) and[MessengerServiceActivities.java](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\MessengerServiceActivities.html) (client) samples.

## Binding to a Service

Application components (clients) can bind to a service by calling [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). The Android system then calls the service's [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) method, which returns an [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) for interacting with the service.

The binding is asynchronous. [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)) returns immediately and does not return the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) to the client. To receive the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html), the client must create an instance of [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html) and pass it to [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). The [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html) includes a callback method that the system calls to deliver the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html).

**Note:** Only activities, services, and content providers can bind to a service—you **cannot** bind to a service from a broadcast receiver.

So, to bind to a service from your client, you must:

1. Implement [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html).

Your implementation must override two callback methods:

[onServiceConnected()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ServiceConnection.html#onServiceConnected(android.content.ComponentName, android.os.IBinder))

The system calls this to deliver the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) returned by the service's [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent)) method.

[onServiceDisconnected()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\ServiceConnection.html#onServiceDisconnected(android.content.ComponentName))

The Android system calls this when the connection to the service is unexpectedly lost, such as when the service has crashed or has been killed. This is not called when the client unbinds.

1. Call [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)), passing the [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html) implementation.
2. When the system calls your [onServiceConnected()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html" \l "onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback method, you can begin making calls to the service, using the methods defined by the interface.
3. To disconnect from the service, call [unbindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "unbindService(android.content.ServiceConnection)).

When your client is destroyed, it will unbind from the service, but you should always unbind when you're done interacting with the service or when your activity pauses so that the service can shutdown while its not being used. (Appropriate times to bind and unbind is discussed more below.)

For example, the following snippet connects the client to the service created above by [extending the Binder class](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\bound-services.html#Binder), so all it must do is cast the returned [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) to the LocalService class and request the LocalService instance:

LocalService mService;  
private ServiceConnection mConnection = new ServiceConnection() {  
    // Called when the connection with the service is established  
    public void onServiceConnected(ComponentName className, IBinder service) {  
        // Because we have bound to an explicit  
        // service that is running in our own process, we can  
        // cast its IBinder to a concrete class and directly access it.  
        LocalBinder binder = (LocalBinder) service;  
        mService = binder.getService();  
        mBound = true;  
    }  
  
    // Called when the connection with the service disconnects unexpectedly  
    public void onServiceDisconnected(ComponentName className) {  
        Log.e(TAG, "onServiceDisconnected");  
        mBound = false;  
    }  
};

With this [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html), the client can bind to a service by passing it to [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)). For example:

Intent intent = new Intent(this, LocalService.class);  
bindService(intent, mConnection, Context.BIND\_AUTO\_CREATE);

* The first parameter of [bindService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "bindService(android.content.Intent, android.content.ServiceConnection, int)) is an [Intent](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Intent.html) that explicitly names the service to bind (thought the intent could be implicit).
* The second parameter is the [ServiceConnection](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html) object.
* **The third parameter is a flag indicating options for the binding. It should usually be**[**BIND\_AUTO\_CREATE**](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#BIND_AUTO_CREATE)**in order to create the service if its not already alive.** Other possible values are [BIND\_DEBUG\_UNBIND](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#BIND_DEBUG_UNBIND) and [BIND\_NOT\_FOREGROUND](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\content\Context.html#BIND_NOT_FOREGROUND), or 0 for none.

### Additional notes

Here are some important notes about binding to a service:

* You should always trap（诱捕） [DeadObjectException](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\DeadObjectException.html) exceptions, which are thrown when the connection has broken. This is the only exception thrown by remote methods.
* Objects are reference counted across processes.
* You should usually pair the binding and unbinding during matching bring-up and tear-down moments of the client's lifecycle. For example:
  + If you only need to interact with the service while your activity is visible, you should bind during [onStart()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Activity.html" \l "onStart()) and unbind during [onStop()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Activity.html" \l "onStop()).
  + If you want your activity to receive responses even while it is stopped in the background, then you can bind during [onCreate()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Activity.html" \l "onCreate(android.os.Bundle)) and unbind during [onDestroy()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Activity.html" \l "onDestroy()). Beware that this implies that your activity needs to use the service the entire time it's running (even in the background), so if the service is in another process, then you increase the weight of the process and it becomes more likely that the system will kill it.

**Note:** You should usually **not** bind and unbind during your activity's [onResume()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Activity.html" \l "onResume()) and [onPause()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Activity.html" \l "onPause()), because these callbacks occur at every lifecycle transition and you should keep the processing that occurs at these transitions to a minimum. Also, if multiple activities in your application bind to the same service and there is a transition between two of those activities, the service may be destroyed and recreated as the current activity unbinds (during pause) before the next one binds (during resume). (This activity transition for how activities coordinate their lifecycles is described in the [Activities](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\guide\components\activities.html#CoordinatingActivities) document.)

For more sample code, showing how to bind to a service, see the [RemoteService.java](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\resources\samples\ApiDemos\src\com\example\android\apis\app\RemoteService.html) class in [ApiDemos](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\resources\\samples\\ApiDemos\\index.html).

## Managing the Lifecycle of a Bound Service

**When a service is unbound from all clients, the Android system destroys it (unless it was also started with [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)))**. As such, you don't have to manage the lifecycle of your service if it's purely a bound service—the Android system manages it for you based on whether it is bound to any clients.

However, if you choose to implement the [onStartCommand()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onStartCommand(android.content.Intent, int, int)) callback method, then you must explicitly stop the service, because the service is now considered to be started. In this case, the service runs until the service stops itself with [stopSelf()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "stopSelf()) or another component calls [stopService()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\Context.html" \l "stopService(android.content.Intent)), regardless of whether it is bound to any clients.

Additionally, if your service is started and accepts binding, then when the system calls your[onUnbind()](file:///E:\android\adt-bundle-windows-x86-20131030\sdk\docs\reference\android\app\Service.html#onUnbind(android.content.Intent)) method, you can optionally return true if you would like to receive a call to [onRebind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onRebind(android.content.Intent))the next time a client binds to the service (instead of receiving a call to [onBind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onBind(android.content.Intent))). [onRebind()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\app\\Service.html" \l "onRebind(android.content.Intent))returns void, but the client still receives the [IBinder](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\os\\IBinder.html) in its [onServiceConnected()](file:///E:\\android\\adt-bundle-windows-x86-20131030\\sdk\\docs\\reference\\android\\content\\ServiceConnection.html" \l "onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback. Below, figure 1 illustrates the logic for this kind of lifecycle.

