Creating and Controlling Services

Services are designed to run in the background, so they need to be started, stopped, and controlled by other application components.

In the following sections, you'll learn how to create a new Service, and how to start and stop it using Intents and the startService method. Later you'll learn how to bind a Service to an Activity, providing a richer interface for interactivity.

Creating a Service

To defi ne a Service, create a new class that extends the Service base class. You'll need to override onBind and onCreate, as shown in the following skeleton class:

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class MyService extends Service {
@Override
public void onCreate() {
// TODO: Actions to perform when service is created.
}
@Override
public IBinder onBind(Intent intent) {
// TODO: Replace with service binding implementation.
return null;
}
}
```

In most cases, you'll also want to override onStart. This is called whenever the Service is started with a call to startService, so it can be executed several times within the Service's lifetime. You should ensure that your Service accounts for this.
The snippet below shows the skeleton code for overriding the onStart method:

```
@Override
public void onStart(Intent intent, int startId) {
// TODO: Actions to perform when service is started.
        }
```

Once you've constructed a new Service, you have to register it in the application manifest. Do this by including a service tag within the application node. You can use attributes on the service tag to enable or disable the Service and specify any permissions required to access it from other applications using a requires-permission flag.
Below is the service tag you'd add for the skeleton Service you created above:

```
<service android:enabled="true" android:name=".MyService"></service>
```

Starting, Controlling, and Interacting with a Service

To start a Service, call startService; you can either implicitly specify a Service to start using an action against which the Service is registered, or you can explicitly specify the Service using its class.

If the Service requires permissions that your application does not have, this call will throw a SecurityException. The snippet below demonstrates both techniques available for starting a Service:

```
// Implicitly start a Service
startService(new Intent(MyService.MY_ACTION));
// Explicitly start a Service
startService(new Intent(this, MyService.class));
```

To use this example, you would need to include a MY_ACTION *property in the* MyService *class and use an Intent Filter to register it as a provider of* MY_ACTION.

To stop a Service, use stopService, passing an Intent that defi nes the Service to stop. This next code snippet fi rst starts and then stops a Service both explicitly and by using the component name returned when calling startService:

```
ComponentName service = startService(new Intent(this, BaseballWatch.class));
// Stop a service using the service name.
stopService(new Intent(this, service.getClass()));
```

```
// Stop a service explicitly.
try {
Class serviceClass = Class.forName(service.getClassName());
stopService(new Intent(this, serviceClass));
} catch (ClassNotFoundException e) {}
```

If startService is called on a Service that's already running, the Service's onStart method will be executed again. Calls to startService do not nest, so a single call to stopService will terminate it no matter how many times startService has been called.