49. Android Local Bound Services – A Worked Example

As outlined in some detail in the previous chapters, bound services, unlike started services, provide a mechanism for implementing communication between an Android service and one or more client components. The objective of this chapter is to build on the overview of bound services provided in *"An Overview of Android Started and Bound Services"* before embarking on an example implementation of a *local* bound service in action.

49.1 Understanding Bound Services

In common with started services, bound services are provided to allow applications to perform tasks in the background. Unlike started services, however, multiple client components may *bind* to a bound service and, once bound, interact with that service using a variety of different mechanisms.

Bound services are created as sub-classes of the Android Service class and must, at a minimum, implement the *onBind()* method. Client components bind to a service via a call to the *bindService()* method. The first bind request to a bound service will result in a call to that service's *onBind()* method (subsequent bind requests do not trigger an *onBind()* call). Clients wishing to bind to a service must also implement a ServiceConnection subclass containing *onServiceConnected()* and *onServiceDisconnected()* methods which will be called once the client-server connection has been established or disconnected, respectively. In the case of the *onServiceConnected()* method, this will be passed an IBinder object containing the information needed by the client to interact with the service.

49.2 Bound Service Interaction Options

There are two recommended mechanisms for implementing interaction between client components and a bound service. In the event that the bound service is local and private to the same application as the client component (in other words it runs within the same process and is not available to components in other applications), the recommended method is to create a subclass of the Binder class and extend it to provide an interface to the service. An instance of this Binder object is then returned by the *onBind()*

method and subsequently used by the client component to directly access methods and data held within the service.

In situations where the bound service is not local to the application (in other words, it is running in a different process from the client component), interaction is best achieved using a Messenger/Handler implementation.

In the remainder of this chapter, an example will be created with the aim of demonstrating the steps involved in creating, starting and interacting with a local, private bound service.

49.3 An Android Studio Local Bound Service Example

The example application created in the remainder of this chapter will consist of a single activity and a bound service. The purpose of the bound service is to obtain the current time from the system and return that information to the activity where it will be displayed to the user. The bound service will be local and private to the same application as the activity.

Launch Android Studio and follow the usual steps to create a new project, entering *LocalBound* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of an Empty Activity named *LocalBoundActivity* with the remaining fields set to the default values.

Once the project has been created, the next step is to add a new class to act as the bound service.

49.4 Adding a Bound Service to the Project

To add a new class to the project, right-click on the package name (located under *app -> java -> com.ebookfrenzy.localbound*) within the Project tool window and select the *New -> Service -> Service* menu option. Specify *BoundService* as the class name and make sure that both the *Exported* and *Enabled* options are selected before clicking on *Finish* to create the class. By default, Android Studio will load the *BoundService.java* file into the editor where it will read as follows:

```
package com.ebookfrenzy.localbound;
```

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

public class BoundService extends Service {
    public BoundService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO: Return the communication channel to the service.
        throw new UnsupportedOperationException("Not yet
implemented");
    }
}
```

49.5 Implementing the Binder

As previously outlined, local bound services can communicate with bound clients by passing an appropriately configured Binder object to the client. This is achieved by creating a Binder subclass within the bound service class and extending it by adding one or more new methods that can be called by the client. In most cases, this simply involves implementing a method that returns a reference to the bound service instance. With a reference to this instance, the client can then access data and call methods within the bound service directly.

For the purposes of this example, therefore, some changes are needed to the template *BoundService* class created in the preceding section. In the first instance, a Binder subclass needs to be declared. This class will contain a single method named *getService()* which will simply return a reference to the current service object instance (represented by the *this* keyword). With these requirements in mind, edit the *BoundService.java* file and modify it as follows:

```
package com.ebookfrenzy.localbound;

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

public class BoundService extends Service {
```

```
    private final IBinder myBinder = new MyLocalBinder();

    public BoundService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO: Return the communication channel to the service.
        throw new UnsupportedOperationException("Not yet
implemented");
    }

    public class MyLocalBinder extends Binder {
        BoundService getService() {
            return BoundService.this;
        }
    }
}
```

Having made the changes to the class, it is worth taking a moment to recap the steps performed here. First, a new subclass of Binder (named *MyLocalBinder*) is declared. This class contains a single method for the sole purpose of returning a reference to the current instance of the *BoundService* class. A new instance of the *MyLocalBinder* class is created and assigned to the *myBinder* IBinder reference (since Binder is a subclass of IBinder there is no type mismatch in this assignment).

Next, the *onBind()* method needs to be modified to return a reference to the *myBinder* object and a new public method implemented to return the current time when called by any clients that bind to the service:

```
package com.ebookfrenzy.localbound;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

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

```
public class BoundService extends Service {

    private final IBinder myBinder = new MyLocalBinder();

    public BoundService() {
    }

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

    public String getCurrentTime() {
        SimpleDateFormat dateformat =
                new SimpleDateFormat("HH:mm:ss MM/dd/yyyy",
                        Locale.US);
        return (dateformat.format(new Date()));
    }

    public class MyLocalBinder extends Binder {
        BoundService getService() {
            return BoundService.this;
        }
    }
}
```

At this point, the bound service is complete and is ready to be added to the project manifest file. Locate and double-click on the *AndroidManifest.xml* file for the *LocalBound* project in the Project tool window and, once loaded into the Manifest Editor, verify that Android Studio has already added a <service> entry for the service as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ebookfrenzy.localbound.localbound" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=" .LocalBoundActivity" >
            <intent-filter>
```

```
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCH
            </intent-filter>
        </activity>

        <service
            android:name=".BoundService"
            android:enabled="true"
            android:exported="true" >
        </service>
    </application>

</manifest>
```

The next phase is to implement the necessary code within the activity to bind to the service and call the *getCurrentTime()* method.

49.6 Binding the Client to the Service

For the purposes of this tutorial, the client is the *LocalBoundActivity* instance of the running application. As previously noted, in order to successfully bind to a service and receive the IBinder object returned by the service's *onBind()* method, it is necessary to create a ServiceConnection subclass and implement *onServiceConnected()* and *onServiceDisconnected()* callback methods. Edit the *LocalBoundActivity.java* file and modify it as follows:

```
package com.ebookfrenzy.localbound;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.IBinder;
import android.content.Context;
import android.content.Intent;
import android.content.ComponentName;
import android.content.ServiceConnection;
import com.ebookfrenzy.localbound.BoundService.MyLocalBinder;

public class LocalBoundActivity extends AppCompatActivity {

    BoundService myService;
    boolean isBound = false;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_local_bound);
    }

    private ServiceConnection myConnection = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName className,
                                       IBinder service) {
            MyLocalBinder binder = (MyLocalBinder) service;
            myService = binder.getService();
            isBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            isBound = false;
        }
    };
}
```

The *onServiceConnected()* method will be called when the client binds successfully to the service. The method is passed as an argument the IBinder object returned by the *onBind()* method of the service. This argument is cast to an object of type MyLocalBinder and then the *getService()* method of the binder object is called to obtain a reference to the service instance, which, in turn, is assigned to *myService*. A Boolean flag is used to indicate that the connection has been successfully established.

The *onServiceDisconnected()* method is called when the connection ends and simply sets the Boolean flag to false.

Having established the connection, the next step is to modify the activity to bind to the service. This involves the creation of an intent and a call to the *bindService()* method, which can be performed in the *onCreate()* method of the activity:

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_local_bound);
        Intent intent = new Intent(this, BoundService.class);
        bindService(intent, myConnection, Context.BIND_AUTO_CREATE);
```

```
}
```

49.7 Completing the Example

All that remains is to implement a mechanism for calling the *getCurrentTime()* method and displaying the result to the user. As is now customary, Android Studio will have created a template *activity_local_bound.xml* file for the activity containing only a TextView. Load this file into the Layout Editor tool and, using Design mode, select the TextView component and change the ID to *myTextView*. Add a Button view beneath the TextView and change the text on the button to read "Show Time", extracting the text to a string resource named *show_time*. On completion of these changes, the layout should resemble that illustrated in Figure 49-1. If any constraints are missing, click on the Infer Constraints button in the Layout Editor toolbar.

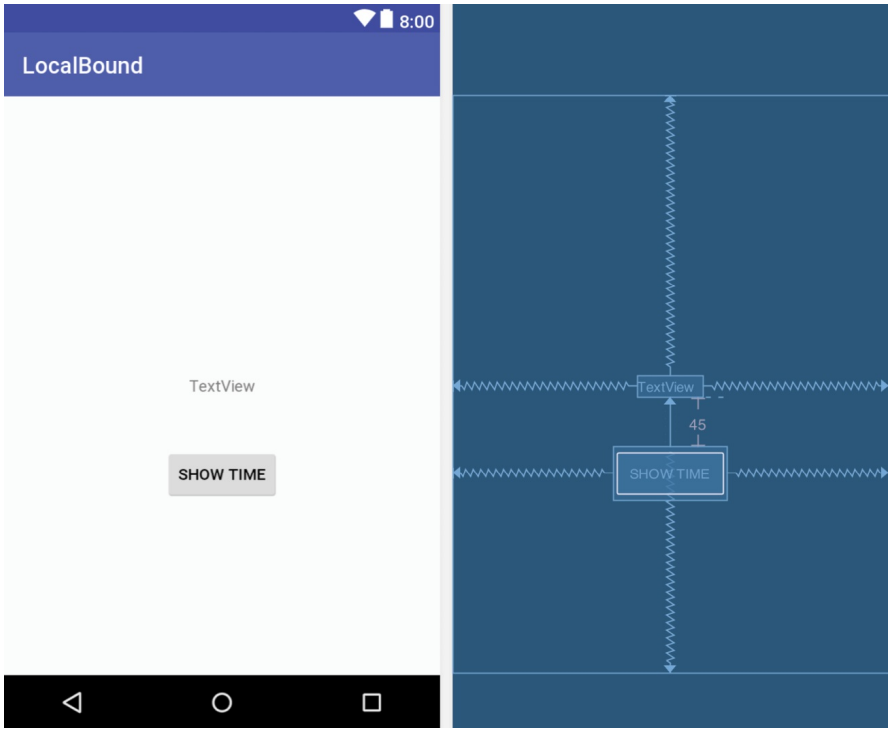

Figure 49-1

Complete the user interface design by selecting the Button and configuring the *onClick* property to call a method named *showTime*.

Finally, edit the code in the *LocalBoundActivity.java* file to implement the *showTime()* method. This method simply calls the *getCurrentTime()* method of the service (which, thanks to the *onServiceConnected()* method, is now

available from within the activity via the *myService* reference) and assigns the resulting string to the TextView:

```
package com.ebookfrenzy.localbound;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.IBinder;
import android.content.Context;
import android.content.Intent;
import android.content.ComponentName;
import android.content.ServiceConnection;
import com.ebookfrenzy.localbound.BoundService.MyLocalBinder;
import android.view.View;
import android.widget.TextView;

public class LocalBoundActivity extends AppCompatActivity {

    BoundService myService;
    boolean isBound = false;

    public void showTime(View view)
    {
        String currentTime = myService.getCurrentTime();
        TextView myTextView =
                (TextView)findViewById(R.id.myTextView);
        myTextView.setText(currentTime);
    }
.
.
.
}
```

49.8 Testing the Application

With the code changes complete, perform a test run of the application. Once visible, touch the button and note that the text view changes to display the current date and time. The example has successfully started and bound to a service and then called a method of that service to cause a task to be performed and results returned to the activity.

49.9 Summary

When a bound service is local and private to an application, components

within that application can interact with the service without the need to resort to inter-process communication (IPC). In general terms, the service's *onBind()* method returns an IBinder object containing a reference to the instance of the running service. The client component implements a ServiceConnection subclass containing callback methods that are called when the service is connected and disconnected. The former method is passed the IBinder object returned by the *onBind()* method allowing public methods within the service to be called.

Having covered the implementation of local bound services, the next chapter will focus on using IPC to interact with remote bound services.