

About This Article

This article provides information about how to use Services in Android applications with AIDL technique. Android application developers can use this article as reference to implement Services in their applications.

Scope:

This article is intended for Android developers wishing to develop mobile applications. It assumes basic knowledge of Android and Java programming languages.

To find out more about Android, please refer to the Knowledge Base under Samsung Developers.

<https://developer.samsung.com/android>

Introduction

A Service is an application component that runs in the background without a user interface. A developer can perform work by using a background service to prepare data for a foreground application. Services work in the background even though the application is running neither in foreground nor background. A service might handle long running tasks like network connections or retrieving database records with the help of content provider from the background.

Service in Android is of two types:

- **Started** - An application component starts the service by calling `startService()` method.
- **Bound** - An application component starts the service by calling `bindService()` method.

Service also has a lifecycle that is like an activity. The major lifecycle callback methods are:

- **onCreate()** - This method gets activated when for the first time the service is loaded into the memory.
- **onStartCommand()** - This method gets activated when any other application component requests for a service to be started, by calling `startService()`.
- **onBind()** - This method gets activated when any other application component wants to bind with the service by calling `bindService()`.
- **onDestroy()** - The system calls this method when the service is no longer used.

[Click here](#) for more details on the Service lifecycle.

AIDL

The Android Interface Definition Language (AIDL) allows developers to define a programming interface that the client and server use to communicate with each other using Inter-Process Communication (IPC).

This article shows how to connect to a running service in Android, and how to retrieve the data from the remote/running service.

Example of IPC mechanism

Let `RemoteService` be a client service and `RemoteServiceClient` be an Activity to communicate with the remote service.

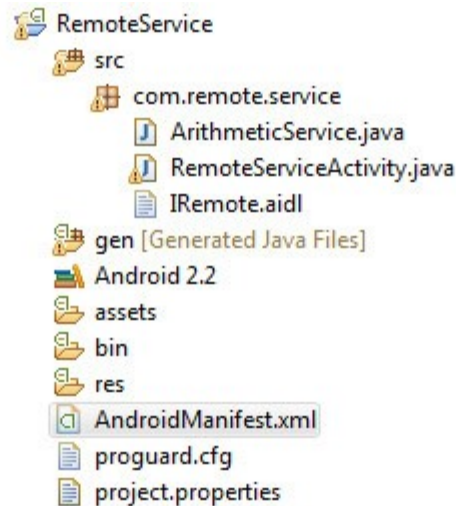
One service provides information about the mathematic operations like addition, subtraction, and multiplication for the given two integers. To expose the functionality of what Service can do, create an **.aidl** file in the project directory.

RemoteService prj

1. Create a new Android project in Eclipse and name it **RemoteService**.
2. Create the following aidl and Service files for interprocess communication:

- An aidl file and name it **IRemote.aidl**.
- Create a class file and name it **ArithmeticService**.

The following figure shows the project structure:



Updating the AIDL

To expose the functionality of the ArithmeticService to clients, update the **IRemote.aidl** file. As discussed before, there are three functions: addition, subtraction, and multiplication. The code snippet in **IRemote.aidl** file is:

```
package com.remote.service;

interface IRemote
{
    int add(int a, int b);
    int subtract(int a, int b);
    double multiply(int a, int b);
}
```

[Click here](#) to learn more about **.aidl** creation and the parameters to be passed

Implementing RemoteService

Once you update the **.aidl** in Eclipse, it automatically generates the remote interface. The remote interface provides a stub inner class which has to have an implementation provided by the ArithmeticService class. The stub class implementation within the service class is as given in the following code snippet:

```
/**
 * IRemote definition is available here
 */
private final IRemote.Stub mBinder = new IRemote.Stub() {
```

```

        @Override
        public int add(int a, int b)
        throws RemoteException {
            // TODO Auto-generated method stub
            return (a + b);
        }
        @Override
        public int subtract(int a, int b)
        throws RemoteException {
            // TODO Auto-generated method stub
            return (a - b);
        }
        @Override
        public double multiply(int a, int b)
        throws RemoteException {
            // TODO Auto-generated method stub
            return (a * b);
        }
    }
};

```

On implementation of a Stub class, use the onBind() method to return the same to the clients. The following code snippet illustrates the use of onBind() method:

```

@Override
    public IBinder onBind(Intent intent)
    {
        // TODO Auto-generated method stub
        return mBinder;
    }

```

Updating the Manifest

Since the service is an application component in Android framework, add a <service> tag in the Android manifest file. The following code snippet shows the <service> tag use:

```

<service android:name=".ArithmeticService">
</service>

```

This is a normal service declaration. However to use the ArithmeticService as a remote service add one more attribute to it:

```

<service android:name=".ArithmeticService" android:process=":remote">
</service>

```

To improve the quality of ArithmeticService, add the <intent-filter> tag to the manifest file. Intent filters are very useful to expose the component to the Android system with some relative data. For more information about the use of intent filters see:

<http://developer.android.com/guide/topics/intents/intents-filters.html#ires>

Now add an action tag to the intent filter. The following code snippet shows the complete manifest file:

```

<?xml version="1.0" encoding="utf-8"?>

```

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.remote.service"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".RemoteServiceActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER"
                />
            </intent-filter>
        </activity>
        <service android:name=".ArithmeticService"
            android:process=":remote">
            <intent-filter >
                <action
android:name="com.remote.service.CALCULATOR"/>
            </intent-filter>
        </service>
    </application>
</manifest>

```

ArithmeticService

```

package com.remote.service;

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

public class ArithmeticService extends Service{
    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return mBinder;
    }
}
/**

```

```

    * IRemote definition is available here
    */
    private final IRemote.Stub mBinder = new IRemote.Stub() {

        @Override
        public int add(int a, int b) throws RemoteException {
            // TODO Auto-generated method stub
            return (a + b);
        }

        @Override
        public int subtract(int a, int b)
            throws RemoteException {
            // TODO Auto-generated method stub
            return (a - b);
        }

        @Override
        public double multiply(int a, int b) throws RemoteException
        {
            // TODO Auto-generated method stub
            return (a * b);
        }

    };
}

```

The next step is to run the application.

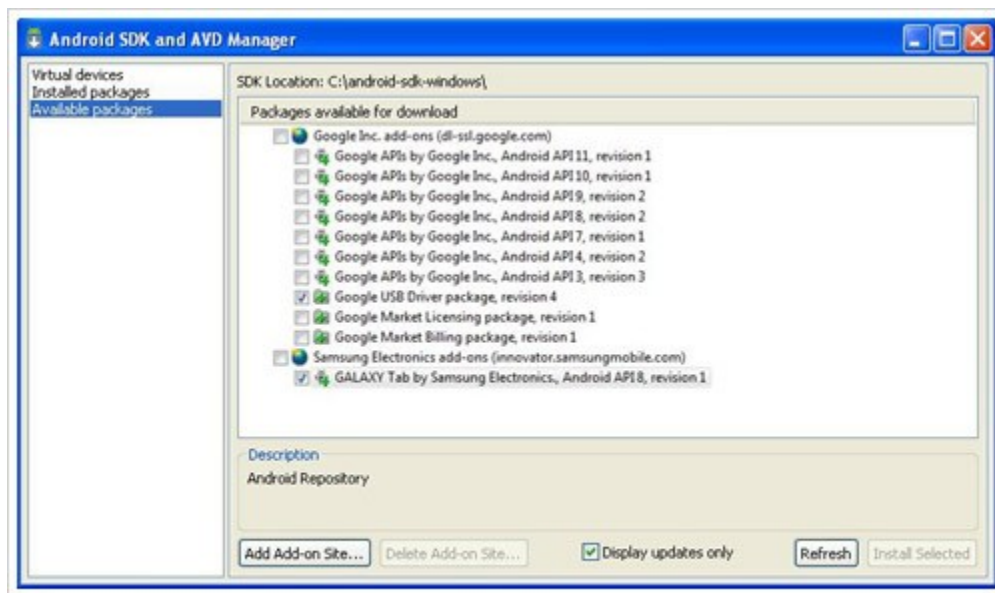


Figure 2: Android SDK and AVD Manager

Here the remote service is created but not running. The remote service gets activated only when any

one client invokes it.

RemoteServiceClient

The implementation of RemoteService provides the mathematic operations to its clients; in similar way create the **RemoteServiceClient** project as a client to RemoteService. To communicate with the service create an **.aidl** file similar to the server that is exposed to it clients under the same package name. The following figure shows the project structure of **RemoteServiceClient**.

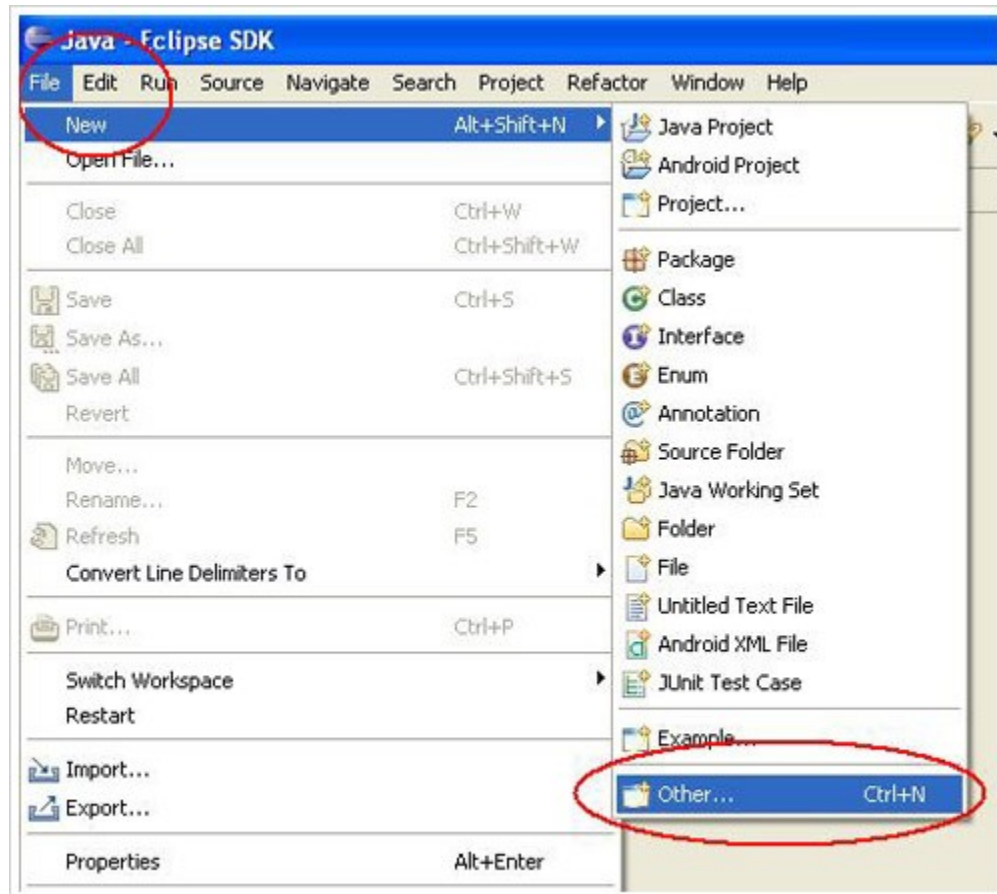


Figure 3: Create New Project

Here the **IRemote.aidl** file is identical to the Service **aidl** file. Now implement UI on the client side creating two edit texts for the input and three buttons for the mathematic operations.

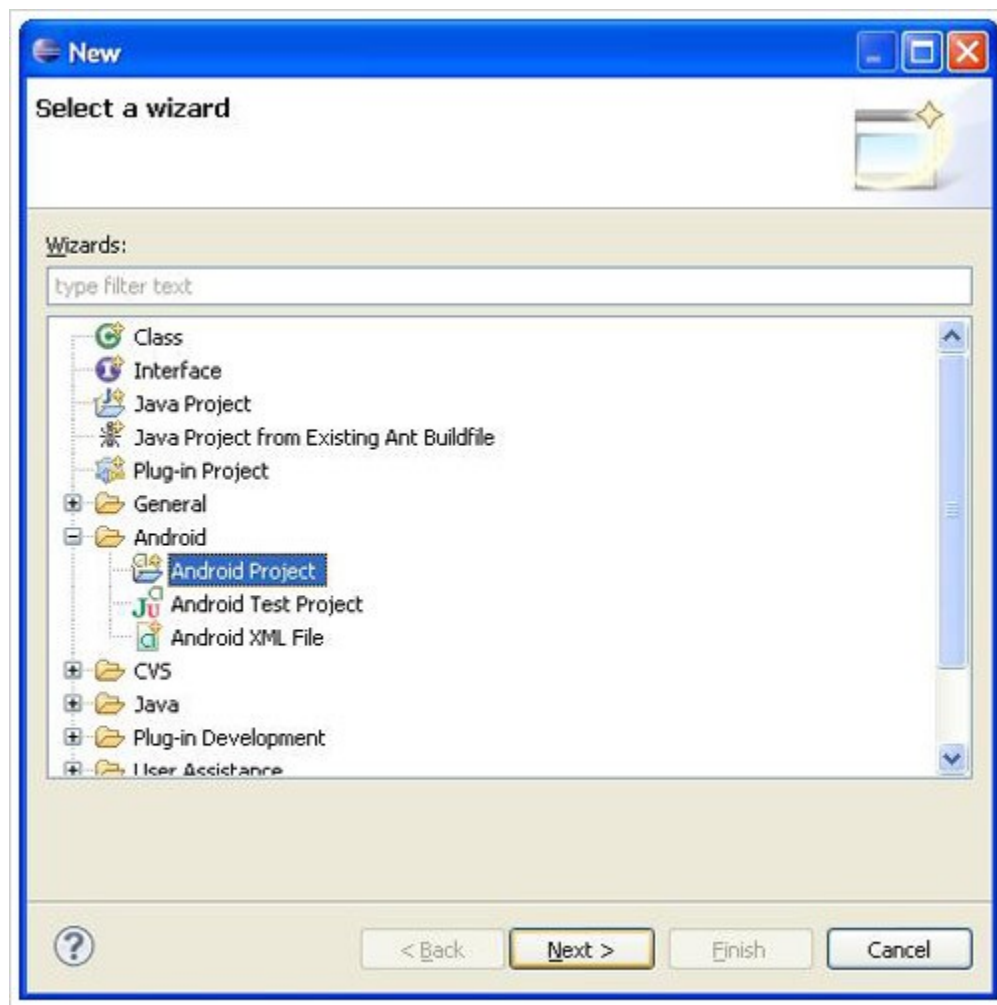


Figure 4: New Project Window

The following .xml file shows the corresponding UI implementation:

```
<?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <TextView android:id="@+id/resultText" android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>

        <EditText android:id="@+id/firstValue" android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:hint="Enter a numeric
value"/>

        <EditText android:id="@+id/secondValue" android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:hint="Enter a numeric
```

```
value"/>
```

```
        <LinearLayout android:layout_width="fill_parent"
android:layout_height="wrap_content"
        android:weightSum="3">

        <Button android:id="@+id/add" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Add"
android:layout_weight="1"/>
        <Button android:id="@+id/subtract" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Subtract"
android:layout_weight="1"/>
        <Button android:id="@+id/multi" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Multiply"
android:layout_weight="1"/>

        </LinearLayout>

</LinearLayout>
```

The next step is to update the RemoteServiceClientActivity class file with the UI functionality as well as the remote service connections. The below code snippet shows the class file with the UI functionality.

```
public class RemoteServiceClientActivity extends Activity implements OnClickListener {
    EditText mFirst,mSecond;
    Button mAdd,mSubtract,mClear;
    TextView mResultText;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mFirst = (EditText) findViewById(R.id.firstValue);
        mSecond = (EditText) findViewById(R.id.secondValue);
        mResultText = (TextView) findViewById(R.id.resultText);

        mAdd = (Button) findViewById(R.id.add);
        mSubtract = (Button) findViewById(R.id.subtract);
        mClear = (Button) findViewById(R.id.multi);

        mAdd.setOnClickListener(this);
        mSubtract.setOnClickListener(this);
        mClear.setOnClickListener(this);
    }
}
```



```

    }
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        switch(v.getId())
        {
            case R.id.add:
            {
            }
            break;
            case R.id.subtract:
            {
            }
            break;
            case R.id.multi:
            {
            }
            break;
        }
    }
}

```

As shown above just the references from the .xml file are taken and added to onclick listeners of each button. Now implement the remote service connections to send the input numbers to the Service as well as to retrieve the result. For creating the connection with the remote service add the following code snippet to the onCreate() method.

```

if(mService == null)
{
    Intent it = new Intent();
    it.setAction("com.remote.service.CALCULATOR");
    //binding to remote service
    bindService(it, mServiceConnection, Service.BIND_AUTO_CREATE);
}

```

Here mService is a reference for the IRemote class. In the bindService() method, the first argument indicates the intent, the second argument indicates the object of the ServiceConnection class, and third argument indicates the flags. The ServiceConnection class is required for the callbacks from the remote service, so it should be implemented in the Activity. The following code snippet shows the implementation of the ServiceConnection class:

```

/**
 * Service connection is used to know the status of the remote service
 */
ServiceConnection mServiceConnection = new ServiceConnection() {

    @Override

```

```

        public void onServiceDisconnected(ComponentName name) {
            // TODO Auto-generated method stub
            mService = null;
            Log.d("IRemote", "Binding - Service disconnected");
        }

        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            // TODO Auto-generated method stub
            mService = IRemote.Stub.asInterface((IBinder) service);
            Log.d("IRemote", "Binding is done - Service connected");
        }
    };

```

If the service binding request has been done successfully, then the callback method `onServiceConnected()` is called. Here `IBinder` should be typecasted to the `IRemote` object. Once the callback method has executed, the client can execute the remote service methods. Now implement the add functionality:

```

case R.id.add:
    {
        int a = Integer.parseInt(mFirst.getText().toString());
        int b = Integer.parseInt(mSecond.getText().toString());

        try {
            mResultText.setText("Result -> Add ->"+mService.add(a,b));
            Log.d("IRemote", "Binding - Add operation");
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}


```

Now implement the functionalities for subtraction and multiplication. Once implemented, run the application and try to execute the functions. The following screen shows the output for addition of given two numbers.

New Android Project

New Android Project

Creates a new Android Project resource.



Project name: HelloWorld

Contents

☒ Create new project in workspace

☐ Create project from existing source

☒ Use default location

Location: C:/eclipse-3.6.2-Classic/workspace/HelloWorld

Browse...

☐ Create project from existing sample

Samples: ApiDemos

Build Target

| Target Name | Vendor | Platform | API... |
|---|---------------------------------------|------------|--------|
| <input checked="" type="checkbox"/> Android 1.5 | Android Open Source Project | 1.5 | 3 |
| <input type="checkbox"/> Google APIs | Google Inc. | 1.5 | 3 |
| <input type="checkbox"/> Android 1.6 | Android Open Source Project | 1.6 | 4 |
| <input type="checkbox"/> Google APIs | Google Inc. | 1.6 | 4 |
| <input type="checkbox"/> Android 2.1-update1 | Android Open Source Project | 2.1-upd... | 7 |
| <input type="checkbox"/> Google APIs | Google Inc. | 2.1-upd... | 7 |
| <input type="checkbox"/> Android 2.2 | Android Open Source Project | 2.2 | 8 |
| <input type="checkbox"/> Google APIs | Google Inc. | 2.2 | 8 |
| <input type="checkbox"/> Real3D Add-On | LGE | 2.2 | 8 |
| <input type="checkbox"/> GALAXY Tab Addon | Samsung Electronics Co., Ltd. | 2.2 | 8 |
| <input type="checkbox"/> Android 2.3.1 | Android Open Source Project | 2.3.1 | 9 |
| <input type="checkbox"/> Google APIs | Google Inc. | 2.3.1 | 9 |
| <input type="checkbox"/> EDK | Sony Ericsson Mobile Communication... | 2.3.1 | 9 |
| <input type="checkbox"/> Android 2.3.3 | Android Open Source Project | 2.3.3 | 10 |
| <input type="checkbox"/> Google APIs | Google Inc. | 2.3.3 | 10 |
| <input type="checkbox"/> Android 3.0 | Android Open Source Project | 3.0 | 11 |
| <input type="checkbox"/> Google APIs | Google Inc. | 3.0 | 11 |

Android + Google APIs

Properties

Application name: HelloWorld

Package name: world.hello

☒ Create Activity: HelloWorld

Min SDK Version: 3

?

< Back

Next >

Finish

Cancel

Finally, call the `unbindService` in the `onDestroy()` method to ensure that client application can no more use the remote service. The following code snippet shows the `onDestroy()` method:

```
protected void onDestroy() {  
  
    super.onDestroy();  
    unbindService(mServiceConnection);  
  
};
```

Note: While working as a client, create the **aidl** file identical to the server **aidl** and place it under the same package name which server has provided to clients. Otherwise you will get the runtime error: `java.lang.SecurityException: Binder invocation to an incorrect interface`

ref: <http://developer.samsung.com/android/technical-docs/Services-with-AIDL-in-Android>