



developerWorks > Technical topics > SOA and web services > Technical library >

# Accessing a JAX-WS web service from Android

## Learn to create a web service client for an Android device

[Deepak Vohra](#), Web Developer

**Summary:** Web services provide function specific services and are especially suited for mobile devices. KSoap2-android project is a SOAP library for the Android platform. In this article we will access a JAX-WS web service for which a WSDL is provided from an Android client. The web service returns a Hello message in response to a request containing a name. We shall create a web service client for Android using the Eclipse ADT plugin and the KSoap2-android library. We will test the web service client in an AVD (Android Virtual Device). The response from sending a request to a Hello web service with name as the request argument is output on an Android virtual device emulator.

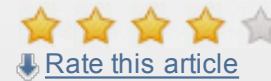
**Date:** 15 Feb 2011

**Level:** Intermediate

**Also available in:** [Korean](#) [Japanese](#) [Portuguese](#)

**Activity:** 65670 views

**Comments:** 19 [View](#) | [Add comment](#) - Sign in



Average rating (110 votes)

## Overview

The article contains the following sections:

1. Pre-requisites
2. Creating a JAX-WS web service
3. Installing the ADT Eclipse Plugin & SDK
4. Creating an Android project
5. Installing the KSoap2-android library
6. Creating the Layout
7. Creating the Activity Class
8. Configuring Internet Access Permission
9. Creating an Android Virtual Device (AVD)
10. Running the Android Client to access the web service

### Table of contents

- Overview
- Pre-requisites
- Creating a JAX-WS web service
- Installing the ADT Eclipse Plugin & SDK
- Creating an Android project
- Installing the Ksoap2-android library
- Creating the layout
- Creating the Activity Class
- Configuring Internet Access Permission
- Creating an AVD
- Running the Android Client to access the web service

## Pre-requisites

The article assumes prior knowledge of web services. We shall create a simple JAX-WS web service with a method that takes a name as an argument and returns a Hello greeting. To develop a web service we would need the following software frameworks.

1. Eclipse IDE
2. An application server that supports JAX-WS web services. For a beginner tutorial on developing a JAX-WS web service with WebSphere refer Resources.

To develop an Android client for the web service we would require the following software components.

1. Eclipse IDE
2. Android SDK
3. ADT Plugin for Eclipse
4. KSoap2-android library

## Creating a JAX-WS web service

A JAX-WS web service essentially consists of a Java class annotated with the `javax.jws.WebService` annotation, the web service endpoint. A web service may optionally consist of a service endpoint interface that is implemented by the service endpoint implementation class. A web service implementation class must not be abstract or `final`. Business methods of the implementation class that are to be exposed as operations to a web service client must be `public`, and must not be `static` or `final`. Create a web service implementation class `HelloWSImpl`, which is annotated with the `@WebService` annotation and implements the `HelloWS` interface. The implementation class contains a method `hello` that takes a `String` parameter for name and returns a Hello message containing the name. The implementation class is listed in Listing 1.

### Listing 1. Web Service Implementation Class HelloWSImpl.java

```
package hello_websevice;
import javax.jws.*;
@WebService(portName = "HelloWSPort", serviceName = "HelloWSService",
    targetNamespace = "http://hello_websevice/",
    endpointInterface = "hello_websevice>HelloWS")
public class HelloWSImpl implements HelloWS {
    public String hello(String name) {
        // replace with your impl here
        return "Hello "+name +" Welcome to Web Services!";
    }
}
```

- Resources
- About the author
- Comments

### Next steps from IBM



Rational Tester for SOA Quality is an automated web services testing and SOA testing tool that supports Web services SOAP, HTTP(S), JMS, WS-Security, UDDI, and RESTful standards.

- Try: The no-charge Rational Service Tester for SOA Quality helps quality assurance professionals validate web service based SOA application..
- Article: See the whats new with Rational Service Tester for SOA Quality, and its Generic Service Client, which provides a single client to interact with any SOA service type.
- Tutorial: Get hands-on experience with this Transport for SOAP feature tutorial using Rational Tester for SOA Quality.
- Buy: Rational Service Tester for SOA Quality

### Dig deeper into SOA and Web services on developerWorks

➔ [Overview](#)

➔ [New to SOA and Web services](#)

➔ [Downloads and products](#)

➔ [Open source projects](#)

```
}  
}
```

The service endpoint interface HelloWS contains the hello method annotated with the @WebMethod annotation and is listed in Listing 2.

#### Listing 2. Service Endpoint Interface HelloWS.java

```
package hello_webservice;  
import javax.jws.WebMethod;  
import javax.jws.WebService;  
@WebService(name = "HelloWS", targetNamespace =  
    "http://hello_webservice/")  
public interface HelloWS {  
    @WebMethod(operationName = "hello")  
    public String hello(String name);  
}
```

Next, generate a WSDL for the web service. The web service shall be made available to an Android client as a WSDL. A WSDL is a document in XML format for describing a web service as a collection of network endpoints, which are also called ports. Messages are abstract definitions of the data being exchanged, and port types are an abstract collections of operations. The Design view of the WSDL for the HelloWSService web service is shown in Figure 1.

Figure 1. Design View of the WSDL

[Standards](#)

[Technical library \(articles, tutorials, and more\)](#)

[Forums](#)

[Events](#)

[New sletter](#)

#### IBM SmartCloud trial. No charge.



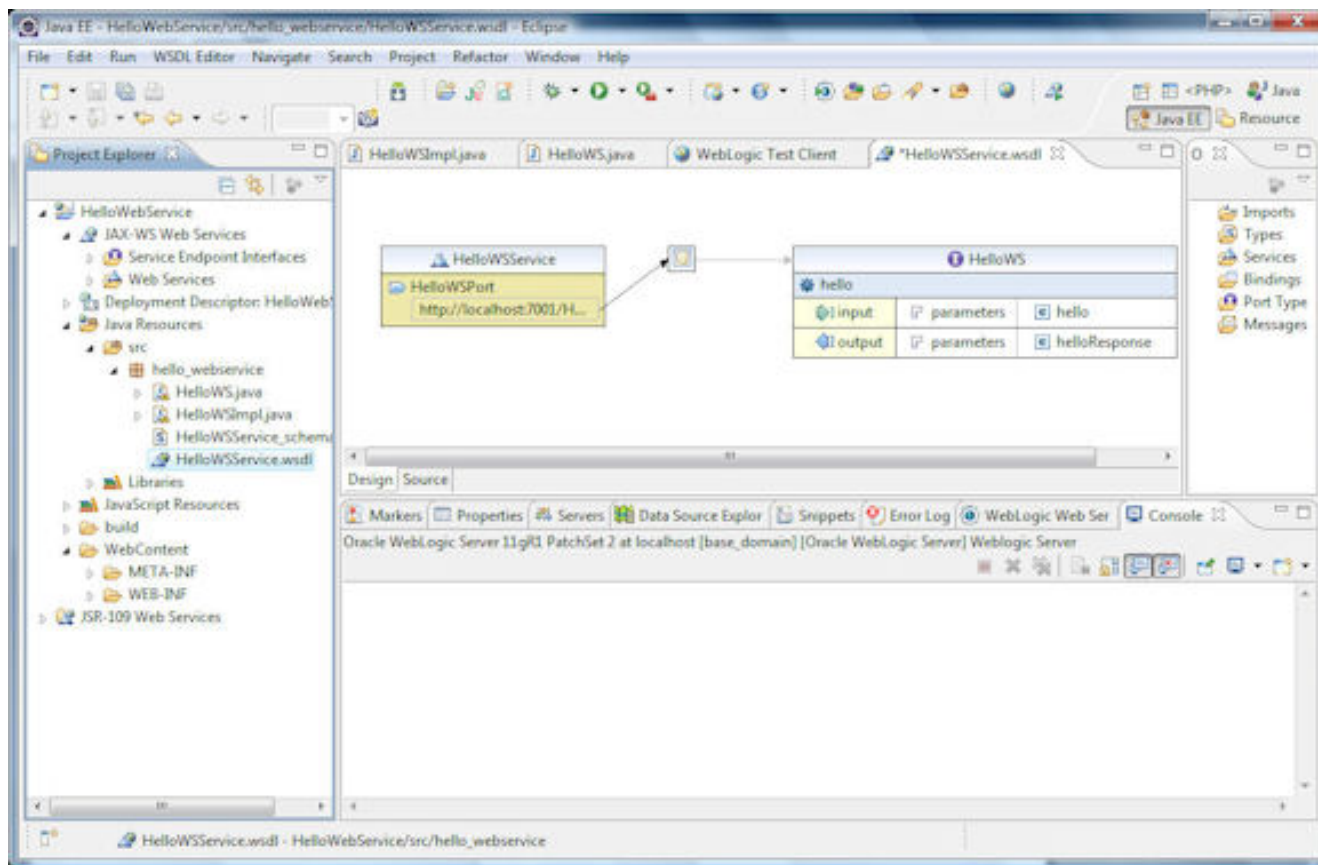
Unleash the power of hybrid cloud computing today!

[Public and private solutions in one trial.](#)

#### Special offers




On demand demos




The WSDL for the web service is shown in Figure 2.


**Figure 2. Web service WSDL**



On demand demos.  
An easy way to  
watch and learn

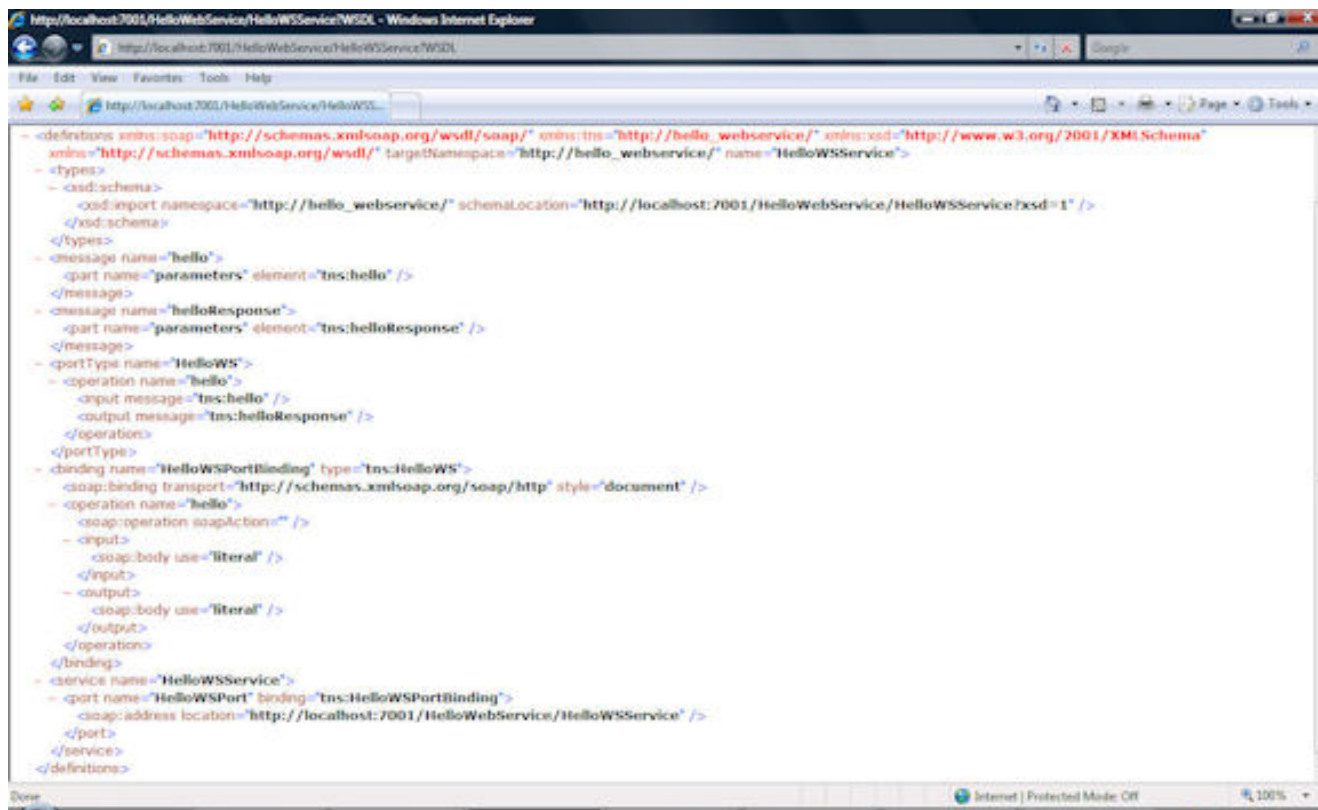


Get recognized!  
dW Author  
Program



Cloud computing  
resources for  
IT professionals

➔ Trial softw are offers

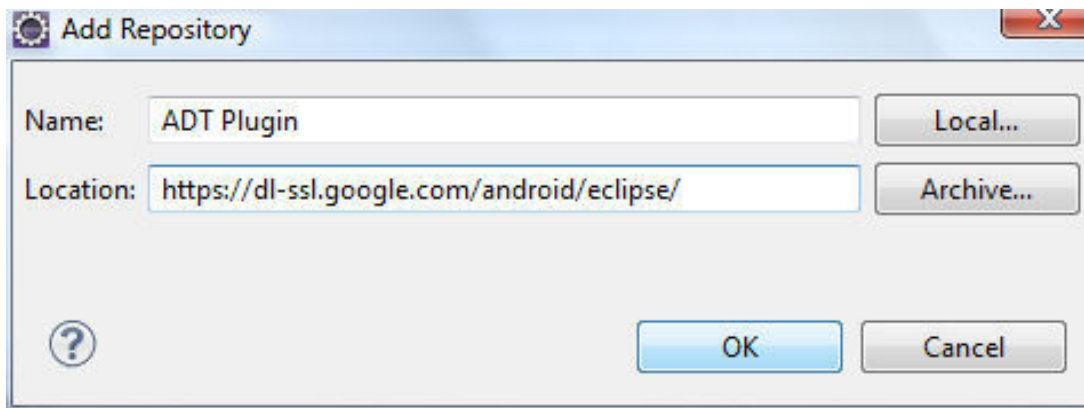


[↑ Back to top](#)

## Installing the ADT Eclipse Plugin & SDK

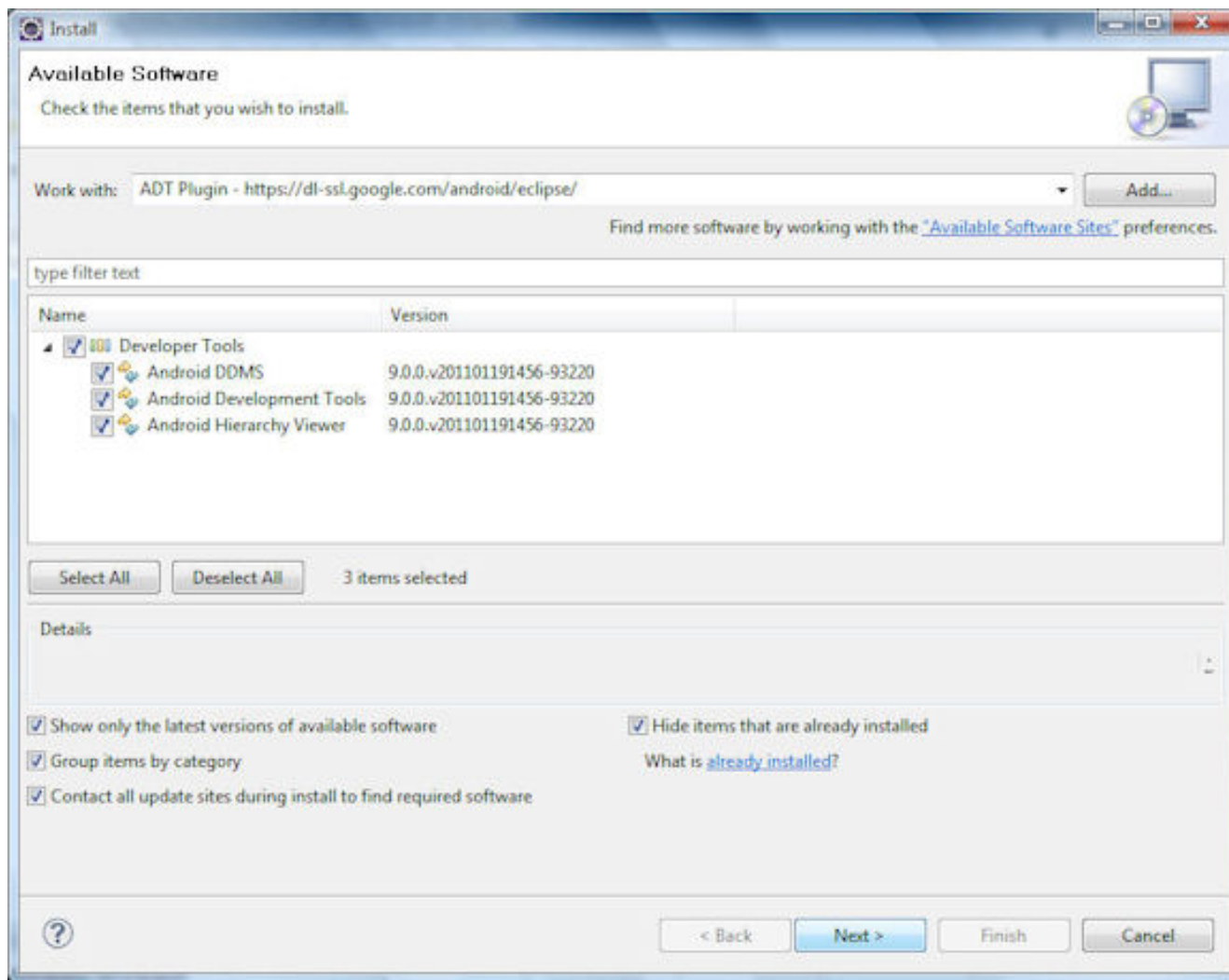
The Android Development Tools (ADT) is a plugin for Eclipse IDE that provides an integrated environment for developing Android applications. Android SDK provides tools and libraries to develop Android applications. To install the ADT Plugin, select Help>Install New Software... Click on Add. In Add Repository specify a Name and specify Location as <https://dl-ssl.google.com/android/eclipse/> as shown in Figure 3. Click on OK.

Figure 3. Creating a Repository for ADT Plugin



Select the Developer Tools listed as shown in Figure 4. and click on Next.

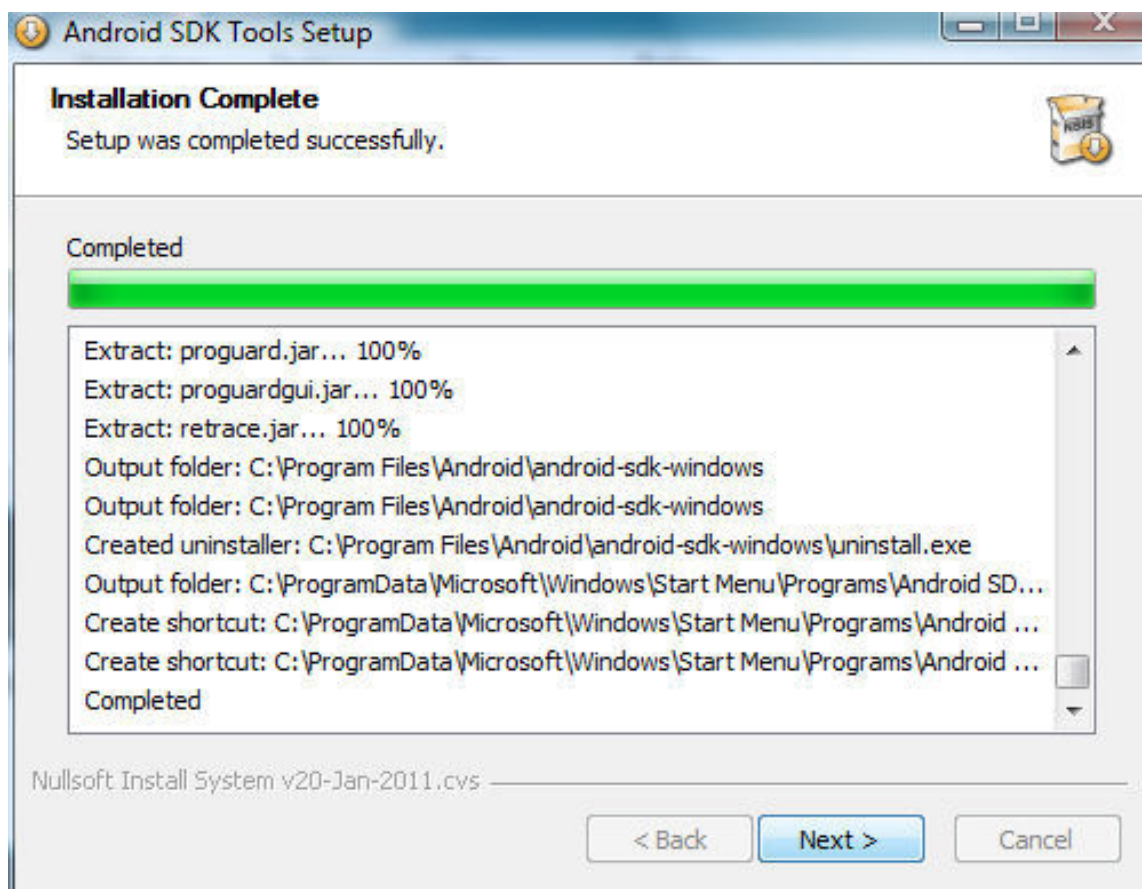
**Figure 4. Installing Developer Tools**



Click on Next in Install Details. Accept the terms of the license agreements and click on Finish. The ADT Plugin gets installed. Next, install the Android SDK. Download the Android starter package (installer\_r09-windows.exe for Windows). Click on the installer to start the Android SDK Tools Setup Wizard. The JDK is required to install the Android SDK; install JDK 5.0/6.0 if not already installed. Click on Next. Specify the Install Location (C:\Program Files\Android\android-sdk-windows is the default) and click on Next. Choose a Start Menu Folder and click on Install. The Android SDK Tools gets installed as shown in Figure 5. Click on Next.

**Figure 5. Installing Android SDK Tools**

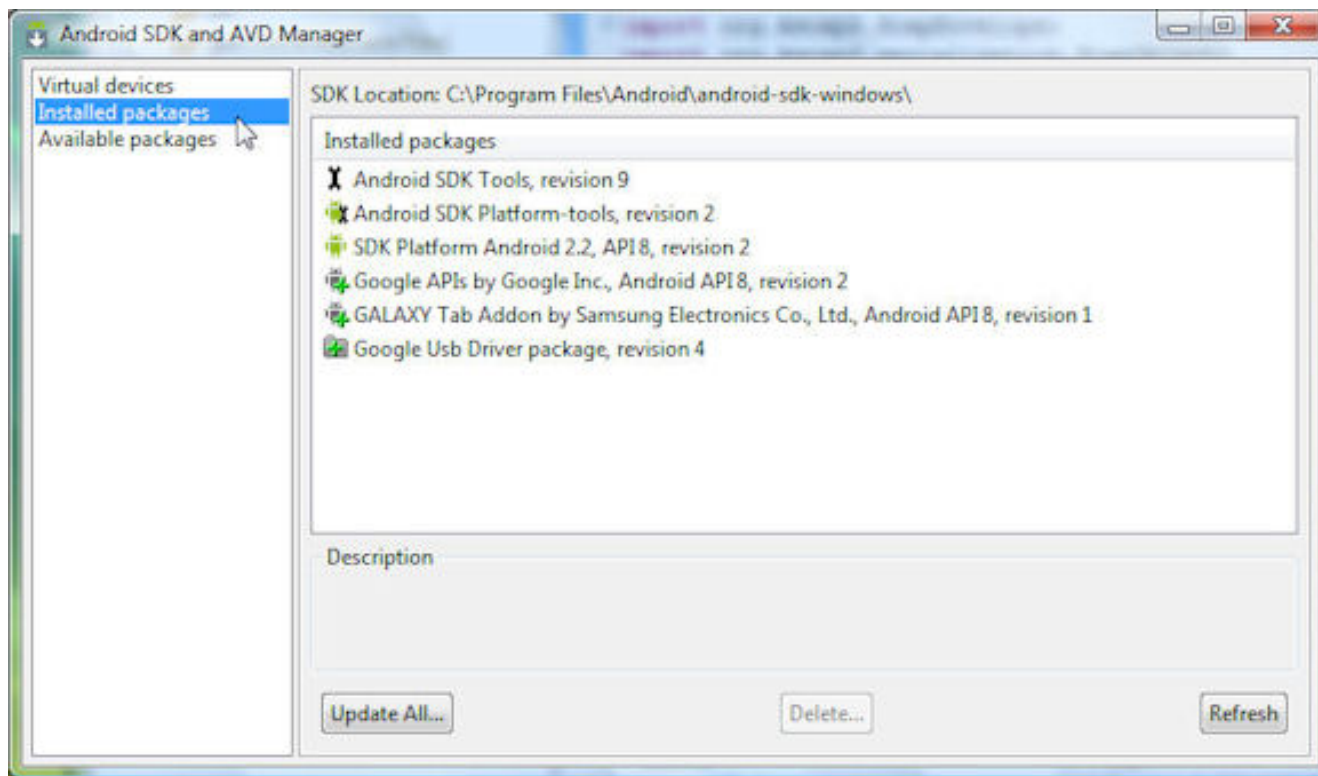




Click on Finish. The Android SDK and AVD Manager gets started. Select packages to install from Available Packages. Install the packages shown in Figure 6.

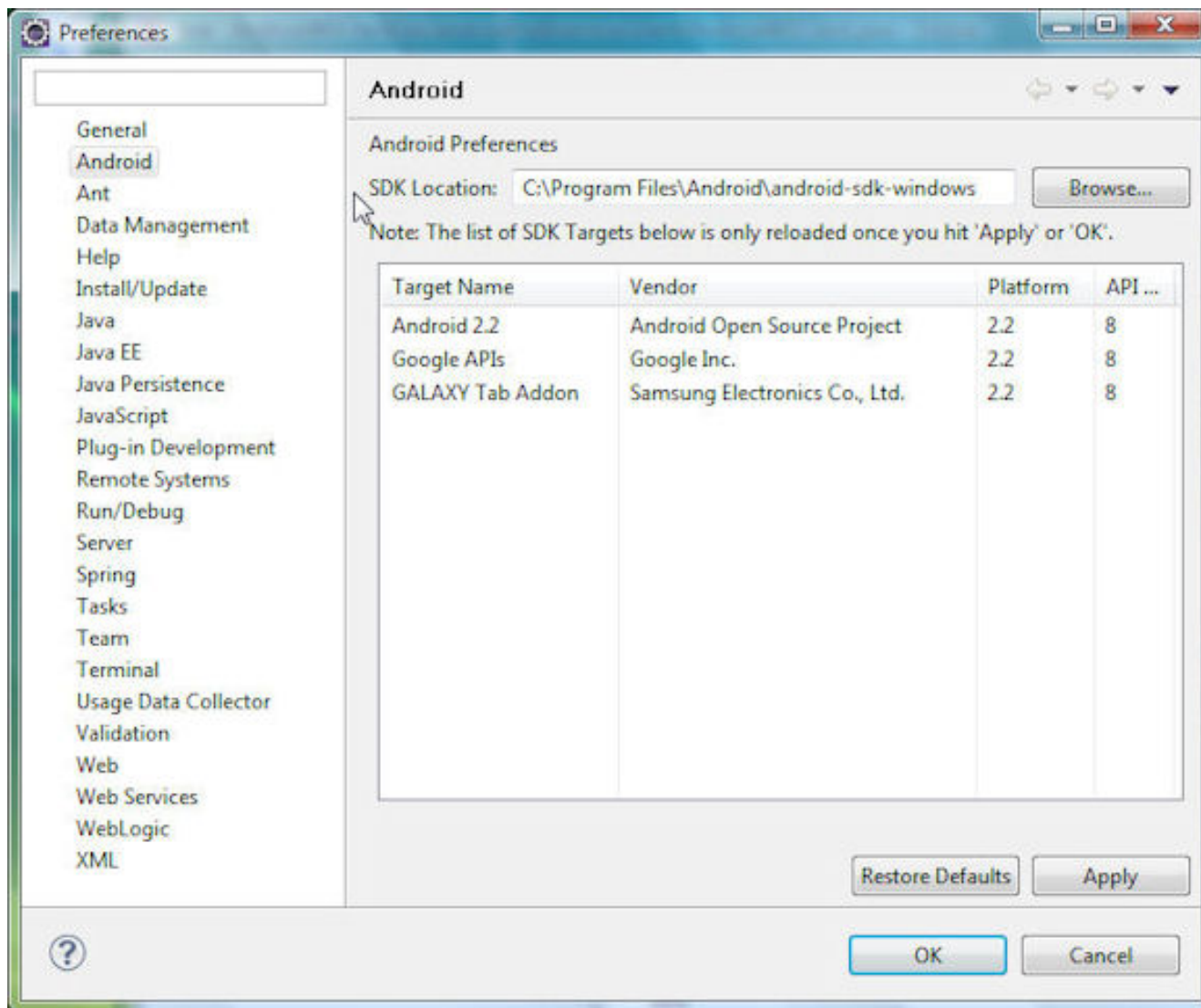
**Figure 6. Android SDK and AVD Manager**





Next, configure the Preferences for Android. Select Window > Preferences in Eclipse. In Preferences select the Android node. Specify the SDK Location as shown in Figure 7.

**Figure 7. Setting Android SDK Location**

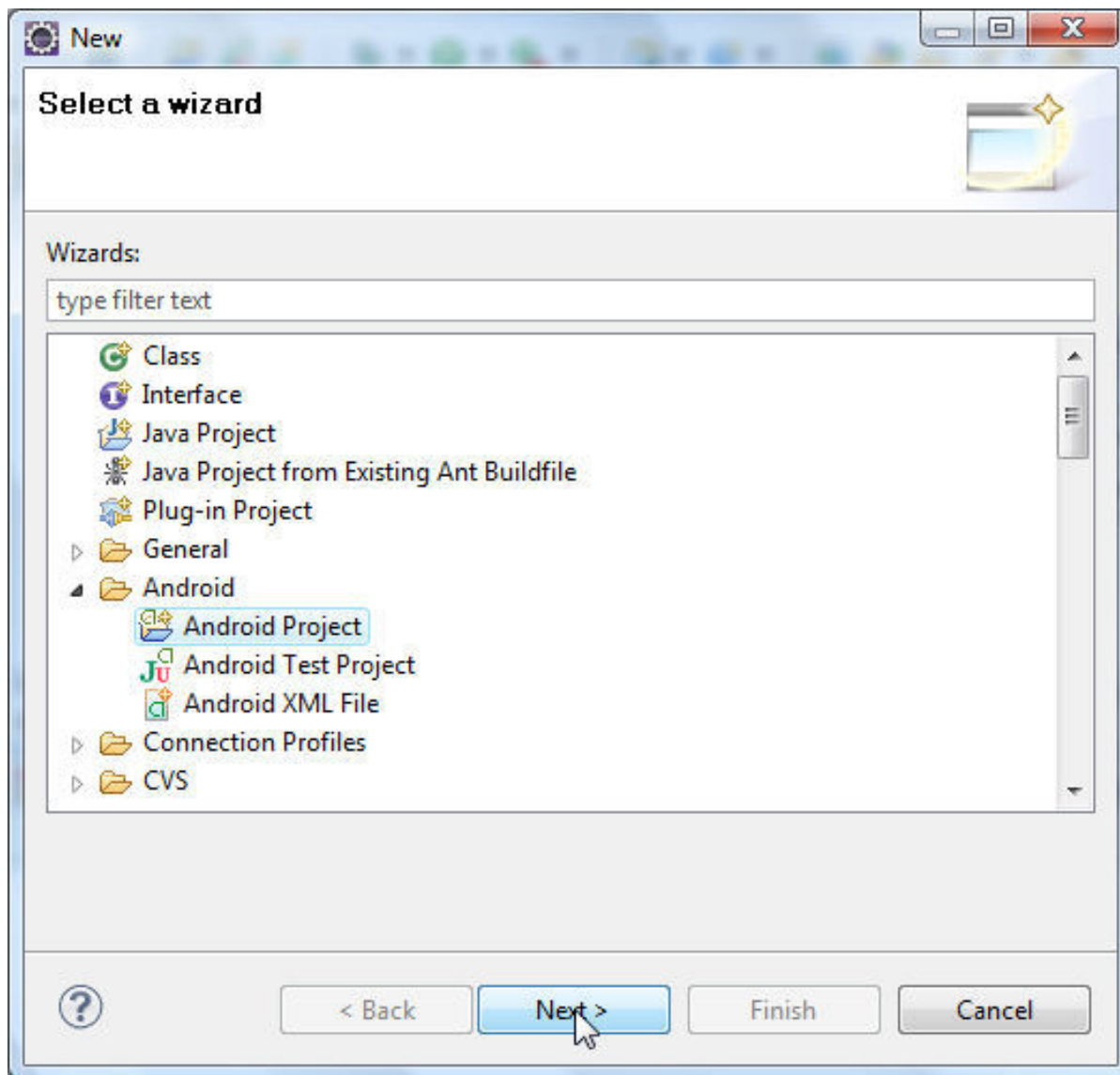


[↑ Back to top](#)

## Creating an Android project

Next, create an Android project in which we shall create the web service client for Android. In Eclipse IDE select File > New. In New select Android>Android Project as shown in Figure 8. Click on Next.

Figure 8. New Android project



In New Android Project specify a Project name (AndroidWSClient). For Build Target select Android 2.2. In Properties specify an Application name, and a package name. Select the checkbox Create Activity and specify the Activity class (AndroidWSClient). An activity represents a user interaction and the class extending the Activity class creates a window for a UI. Specify the minimum SDK version as 8 and click on Next as shown in Figure 9.

**Figure 9. Creating a new Android project**



## New Android Project



Creates a new Android Project resource.

Project name:

### Contents

- ☒ Create new project in workspace  
☐ Create project from existing source  
☒ Use default location

Location:

- ☐ Create project from existing sample

Samples:

### Build Target

Target Name	Vendor	Platform	API ...
<input checked="" 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"/> GALAXY Tab Add...	Samsung Electronics Co., Ltd.	2.2	8

Standard Android platform 2.2

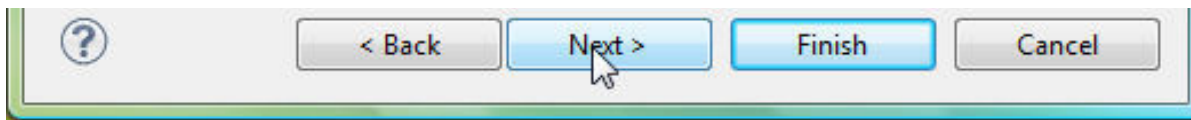
### Properties

Application name:

Package name:

☒ Create Activity:

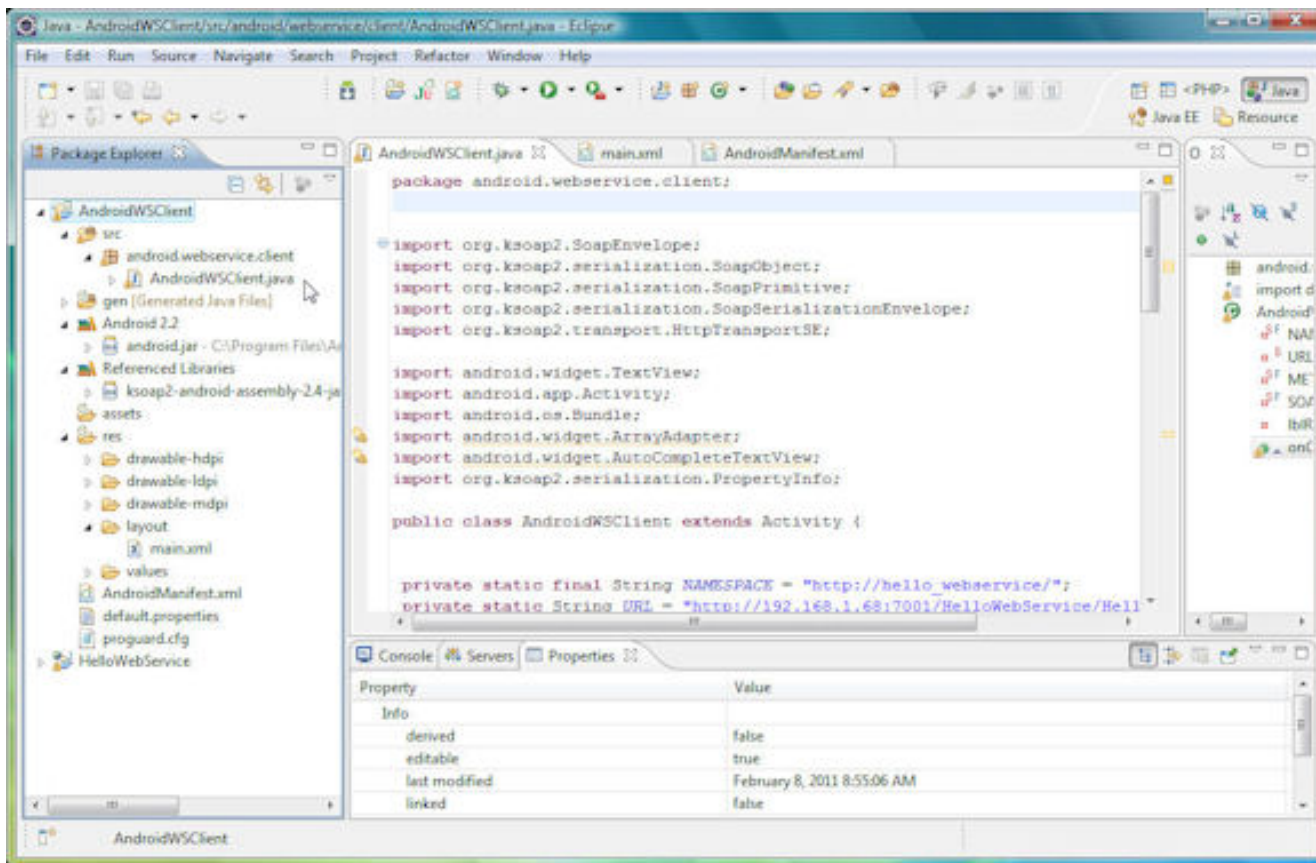
Min SDK Version:



In New Android Test Project a Test Project may be created. We won't be creating a test project. Click on Finish. An Android project gets created as shown in Figure 10. The Android project consists of the following essential components.

1. An activity class(AndroidWSEClient), which extends the Activity class.
2. A res/layout/main.xml file to specify the layout of the Android application.
3. An AndroidManifest.xml file, which contains essential information about the application such as the package name, application components, processes, permissions, and the minimum API level for the Android system.

**Figure 10. Android project**



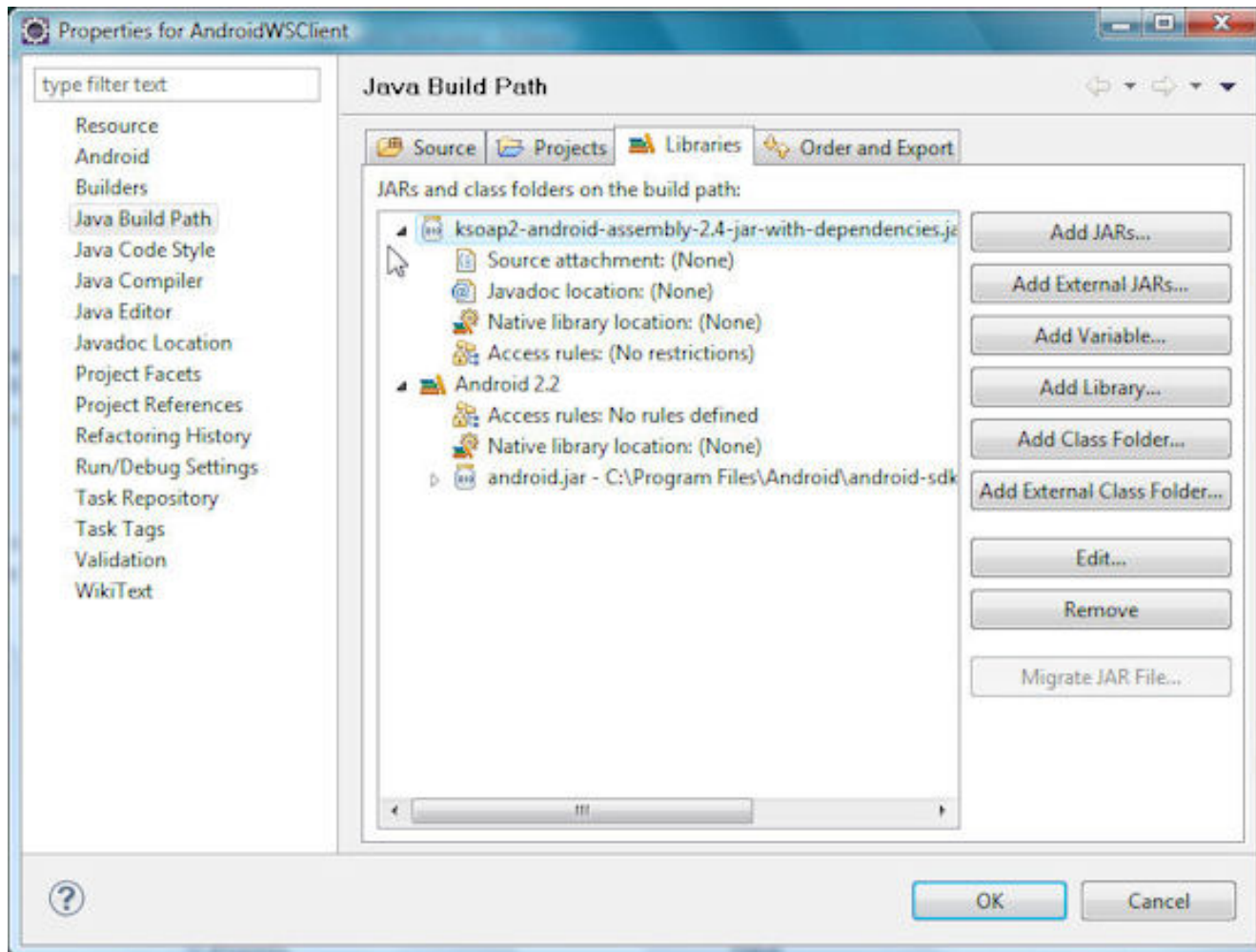
[Back to top](#)



## Installing the Ksoap2-android library

The Ksoap2-android library is a lightweight and efficient SOAP library for the Android platform. Download ksoap2-android-assembly-2.4-jar-with-dependencies.jar from ksoap2-android project. Add the Ksoap2-android assembly JAR, which includes the dependencies to the Java Build Path as shown in Figure 11.

Figure 11. The Ksoap2-android Library



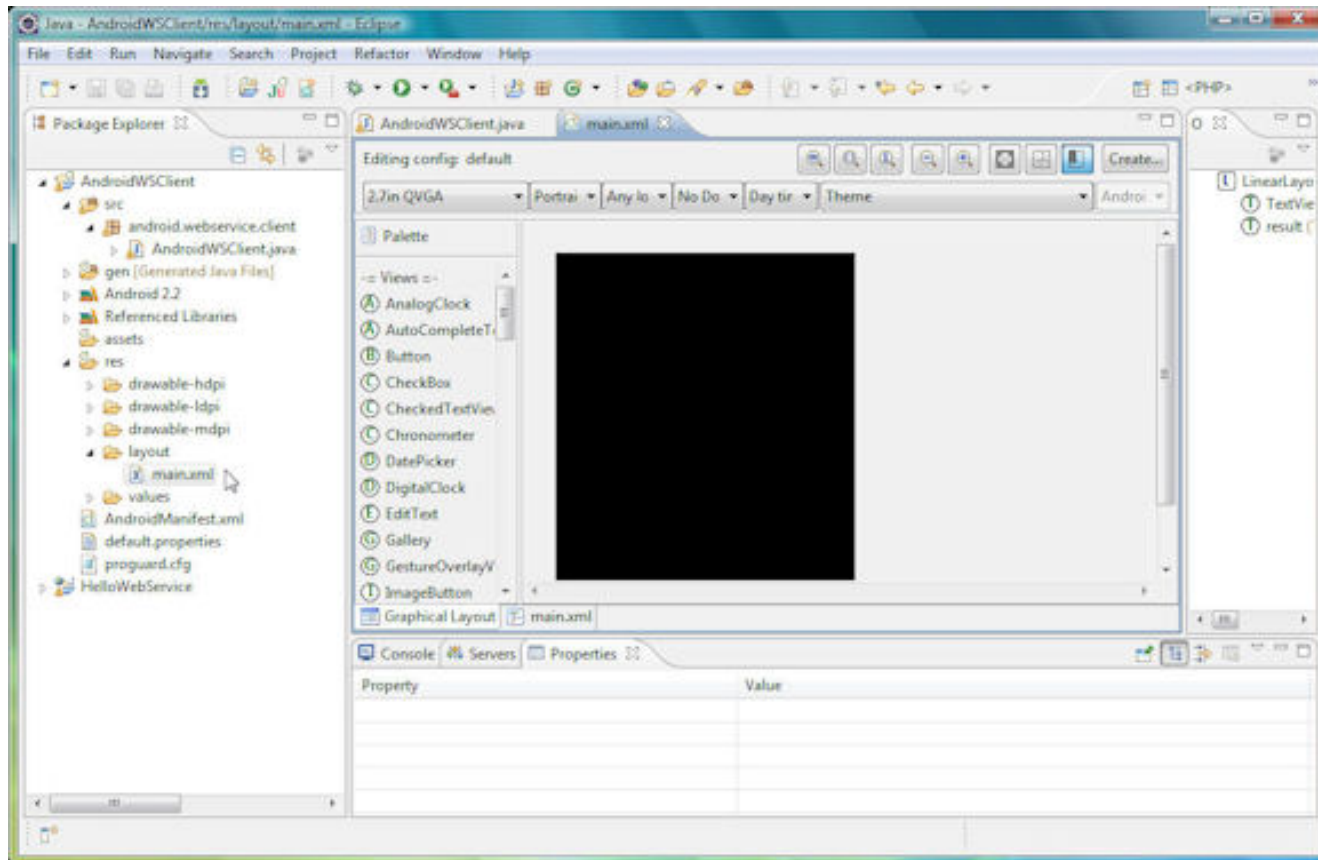
[↑ Back to top](#)

## Creating the layout

In the res/layout/main.xml specify the layout of the Android UI components. We shall create a UI in which the response from

the web service is displayed as a text message. Create a LinearLayout in which sub-elements are displayed in a linear layout, vertically. Add a TextView element, with id result, to display the web service response for a method call to the hello method. The method invocation sends a name as an argument and gets a Hello message as a response. Optionally, create a TextView element for a title. The Graphical Layout of the Android web service-client application is shown in Figure 12. The Android screen is essentially blank with a provision to display a web service response as a text message.

**Figure 12. The Graphical Layout**



The main.xml layout file is listed in Listing 3.

**Listing 3. Layout file main.xml**

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



```
        android:layout_height="wrap_content" android:text="" />
<TextView android:id="@+id/result" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

[↑ Back to top](#)

## Creating the Activity Class

The `AndroidWClient` class is the activity class of an Android application and extends the `Activity` class. In this section we shall construct the `AndroidWClient` activity class. Import the required Ksoap2-android classes:

```
import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapPrimitive;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import org.ksoap2.serialization.PropertyInfo;
```

Define constants discussed in Table 1.

**Table 1. Constants in Activity Class**

Constant	Description
NAMESPACE	Namespace is the targetNamespace in the WSDL.
URL	The WSDL URL. Its value is the location attribute of the soap:address element for a port element in a WSDL. Unless the web service is also hosted on the Android device, the hostname should not be specified as localhost, because the application runs on the Android device while the web service is hosted on the localhost server. Specify hostname as the IP address of the server hosting the web service.
METHOD_NAME	The name of the web service operation, which may be obtained from the WSDL.
SOAP_ACTION	NAMESPACE+METHOD_NAME specified as a String literal.

The constant values are as follows; the IP address for the URL constant would vary:

```
private static final String NAMESPACE = "http://hello_webservice/";
private static String URL="http://192.168.1.68:7001/HelloWebService/HelloWSService?WSDL";
private static final String METHOD_NAME = "hello";
private static final String SOAP_ACTION = "http://hello_webservice/hello";
```

All Activities must implement the onCreate method for activity initialization. Define the UI using the setContentView method and the layout resource.

```
setContentView(R.layout.main);
```

Create an Android widget TextView object using the findViewById method on the TextView element defined in the main.xml.

```
TextView lblResult = (TextView) findViewById(R.id.result);
```

Create a org.ksoap2.serialization.SoapObject object to build a SOAP request. Specify the namespace of the SOAP object and method name to be invoked in the SoapObject constructor.

```
SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
```

Create a org.ksoap2.serialization.PropertyInfo object to contain property information to be sent with the SOAP method call. Each property requires a new PropertyInfo object. The hello method takes only 1 argument for a name. Set the property name as "arg0", and specify the type of the property as STRING\_CLASS. Add the PropertyInfo object to the SoapObject using the addProperty method.

```
PropertyInfo propInfo=new PropertyInfo();  
propInfo.name="arg0";  
propInfo.type=PropertyInfo.STRING_CLASS;  
request.addProperty(propInfo, "John Smith");
```

Next create a SOAP envelop. Use the SoapSerializationEnvelope class, which extends the SoapEnvelope class, with support for SOAP Serialization format, which represents the structure of a SOAP serialized message. The main advantage of SOAP serialization is portability.

```
SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
```

The constant SoapEnvelope.VER11 indicates SOAP Version 1.1. Assign the SoapObject request object to the envelop as the outbound message for the SOAP method call.

```
envelope.setOutputSoapObject(request);
```

Create a org.ksoap2.transport.HttpTransportSE object that represents a J2SE based HttpTransport layer. HttpTransportSE extends the org.ksoap2.transport.Transport class, which encapsulates the serialization and deserialization of SOAP messages.

```
HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);
```

Make the soap call using the SOAP\_ACTION and the soap envelop.

```
androidHttpTransport.call(SOAP_ACTION, envelope);
```

Get the web service response using the getResponse method of the SoapSerializationEnvelope object and cast the response object to SoapPrimitive, class used to encapsulate primitive types.

```
SoapPrimitive resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();
```

Set the String message in the SOAP response in the TextView UI component.

```
lblResult.setText(resultsRequestSOAP.toString());
```

The Activity class is listed below in Listing 4.

#### **Listing 4. Activity Class AndroidWSClient.java**

```
package android.webservice.client;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapPrimitive;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import org.ksoap2.serialization.PropertyInfo;

import android.widget.TextView;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class AndroidWSClient extends Activity {

    private static final String NAMESPACE = "http://hello_webservice/";
    private static String URL = "http://192.168.1.68:7001/HelloWebService/HelloWSService?WSDL";
    private static final String METHOD_NAME = "hello";
    private static final String SOAP_ACTION = "http://hello_webservice/hello";

    private TextView lblResult;
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    lblResult = (TextView) findViewById(R.id.result);

    SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);

    PropertyInfo propInfo=new PropertyInfo();
    propInfo.name="arg0";
    propInfo.type=PropertyInfo.STRING_CLASS;

    request.addProperty(propInfo, "John Smith");

    SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);

    envelope.setOutputSoapObject(request);
    HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

    try {
        androidHttpTransport.call(SOAP_ACTION, envelope);

        SoapPrimitive resultsRequestSOAP = (SoapPrimitive) envelope.getResponse();

        lblResult.setText(resultsRequestSOAP.toString());

    } catch (Exception e) {

    }

}
}
}

```

[↑ Back to top](#)

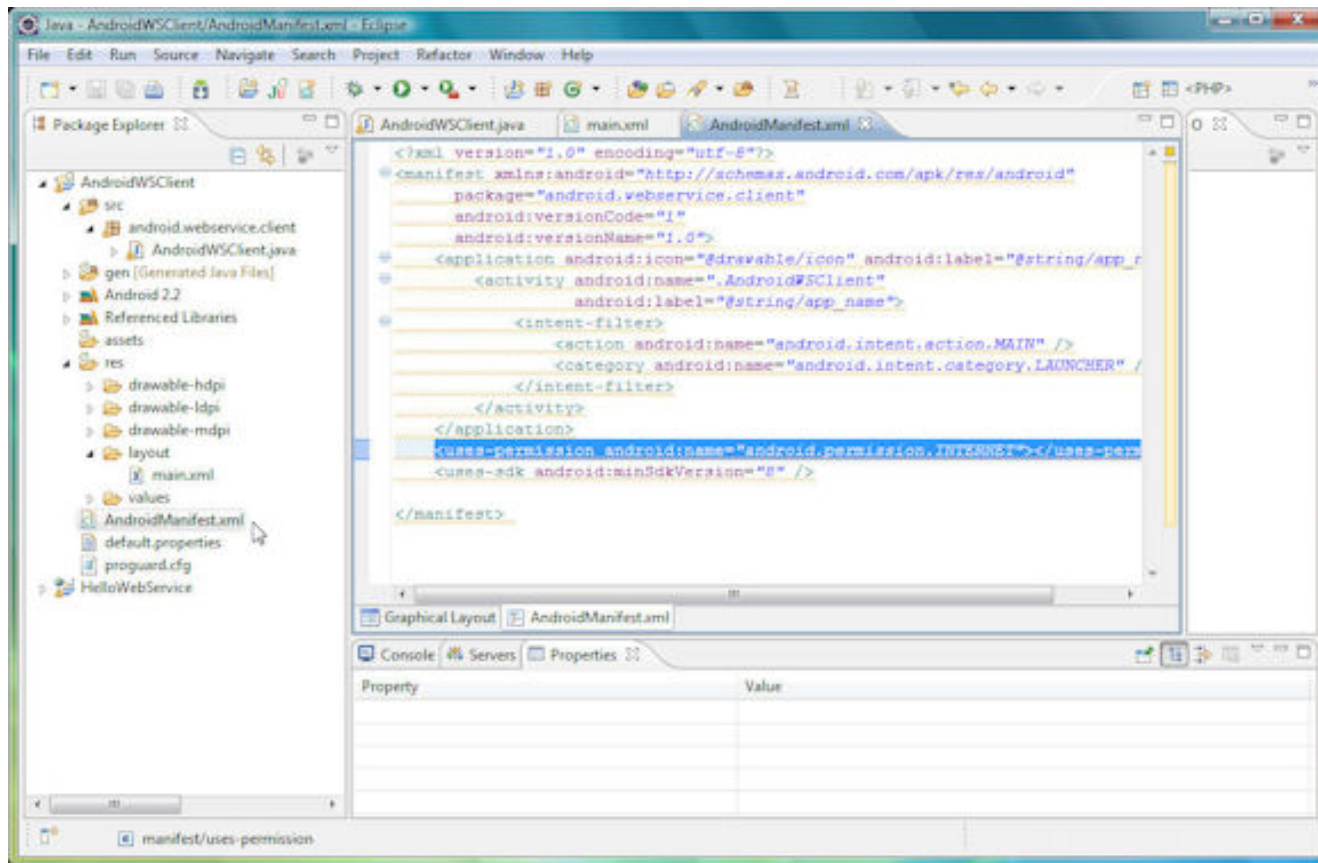
## Configuring Internet Access Permission

To access a web service from an Android device we need to enable a permission in `AndroidManifest.xml` that allows applications to open network sockets. Add the following `uses-permission` element.

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

The AndroidManifest.xml with the permission set is shown in Figure 13.

**Figure 13. Setting the INTERNET Permission**



The AndroidManifest.xml is listed in Listing 5.

**Listing 5. AndroidManifest.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.webservice.client"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".AndroidWSCClient"
            android:label="@string/app_name">
            <intent-filter>
```

```
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-sdk android:minSdkVersion="8" />

</manifest>
</uses-permission>
```

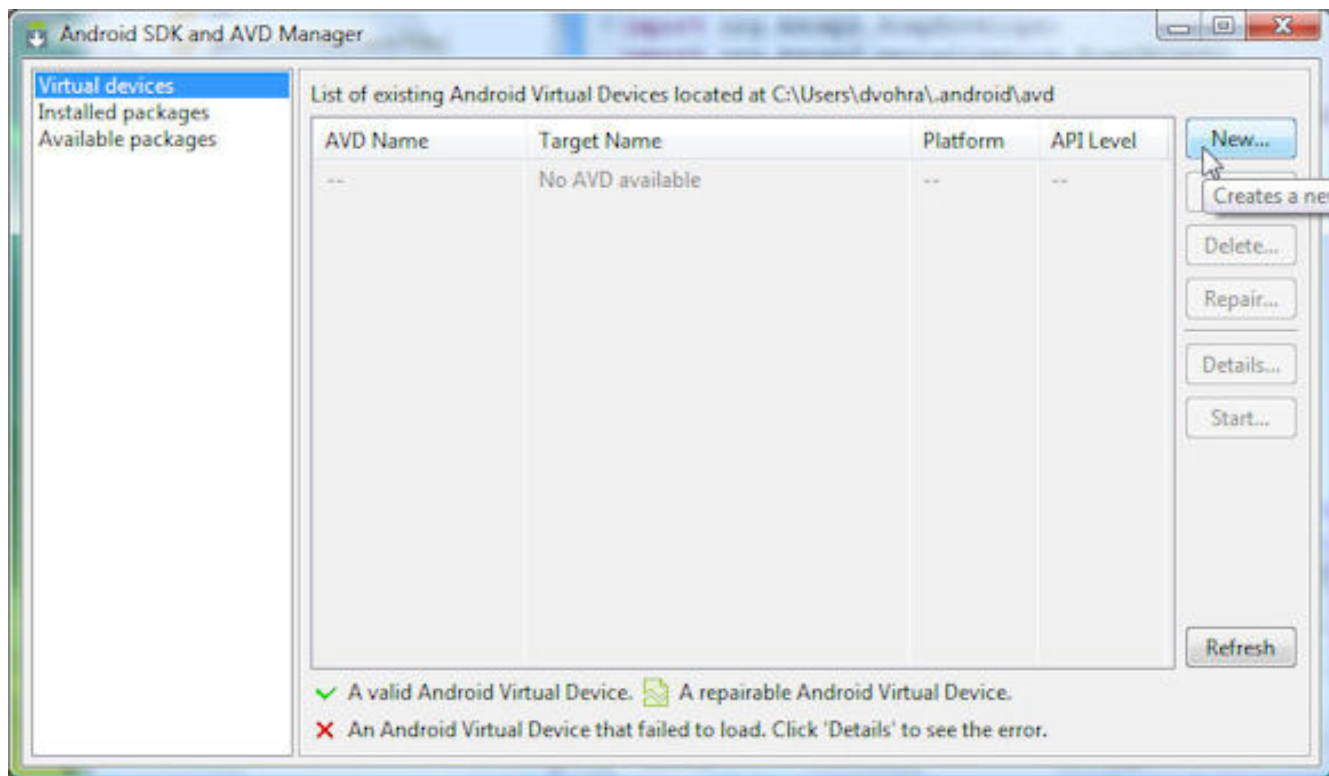
---

 [Back to top](#)

## Creating an AVD

Android SDK provides a mobile device emulator, a virtual mobile device to develop and test Android applications. The emulator supports Android Virtual Device (AVD) configurations with which other virtual mobile device options such as Android platform may be specified. To create an AVD click on Window>Android SDK and AVD Manager. Click on Virtual devices and click on New as shown in Figure 14.

**Figure 14. Creating an AVD**



In Create New AVD specify a Name, select Target as Android 2.2-API level 8. Specify SD Card size as 100 MiB, which should be sufficient to install the web client application. Specifying a lower value such as 10 may generate a "No space left on device" error. Click on Create AVD to create an AVD as shown in Figure 15.

**Figure 15. Configuring AVD**



Create new Android Virtual Device (AVD)

Name:

Target:

SD Card:

☒ Size:

☐ File:

Snapshot:

☐ Enabled

Skin:

☒ Built-in:

☐ Resolution:  x

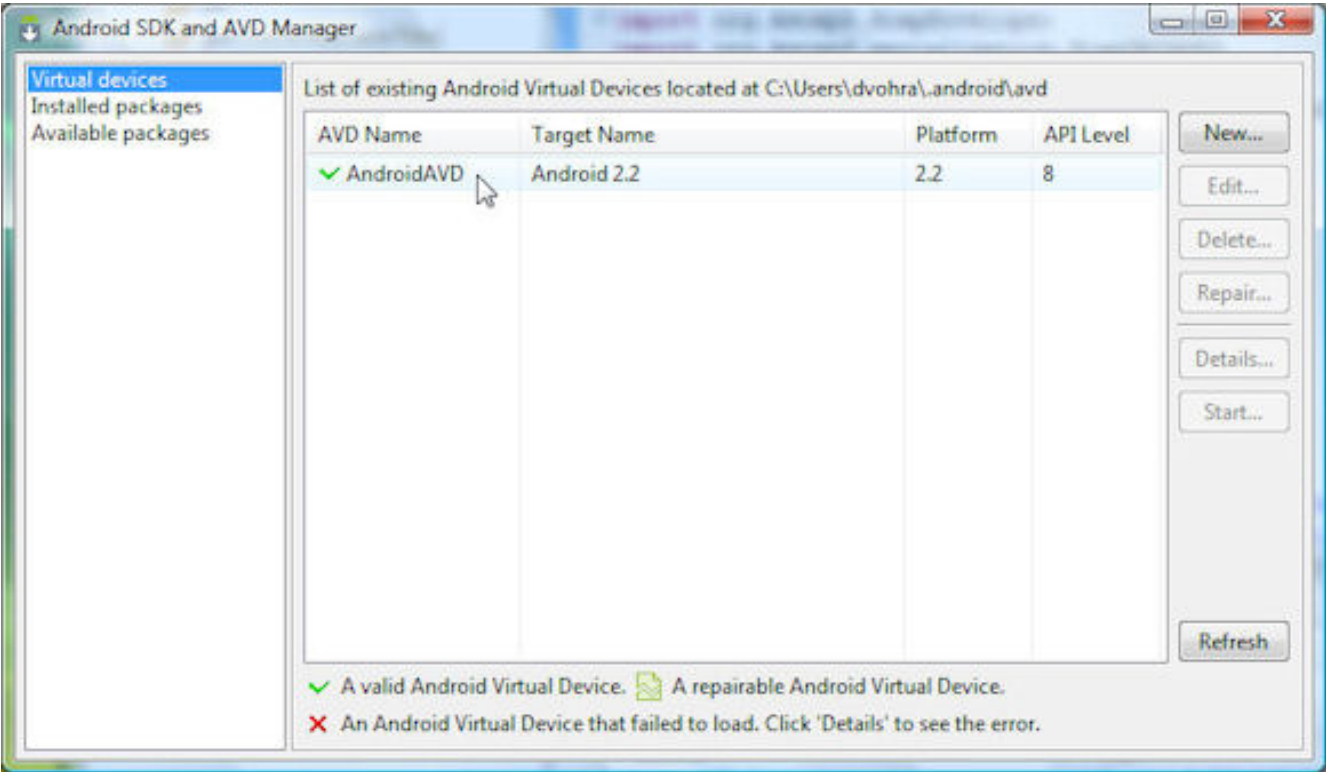
Hardware:

Property	Value	
Abstracted LCD density	240	
Max VM application hea...	24	

☐ Override the existing AVD with the same name

A new AVD get created as shown in Figure 16.

Figure 16. New AVD



[↑ Back to top](#)

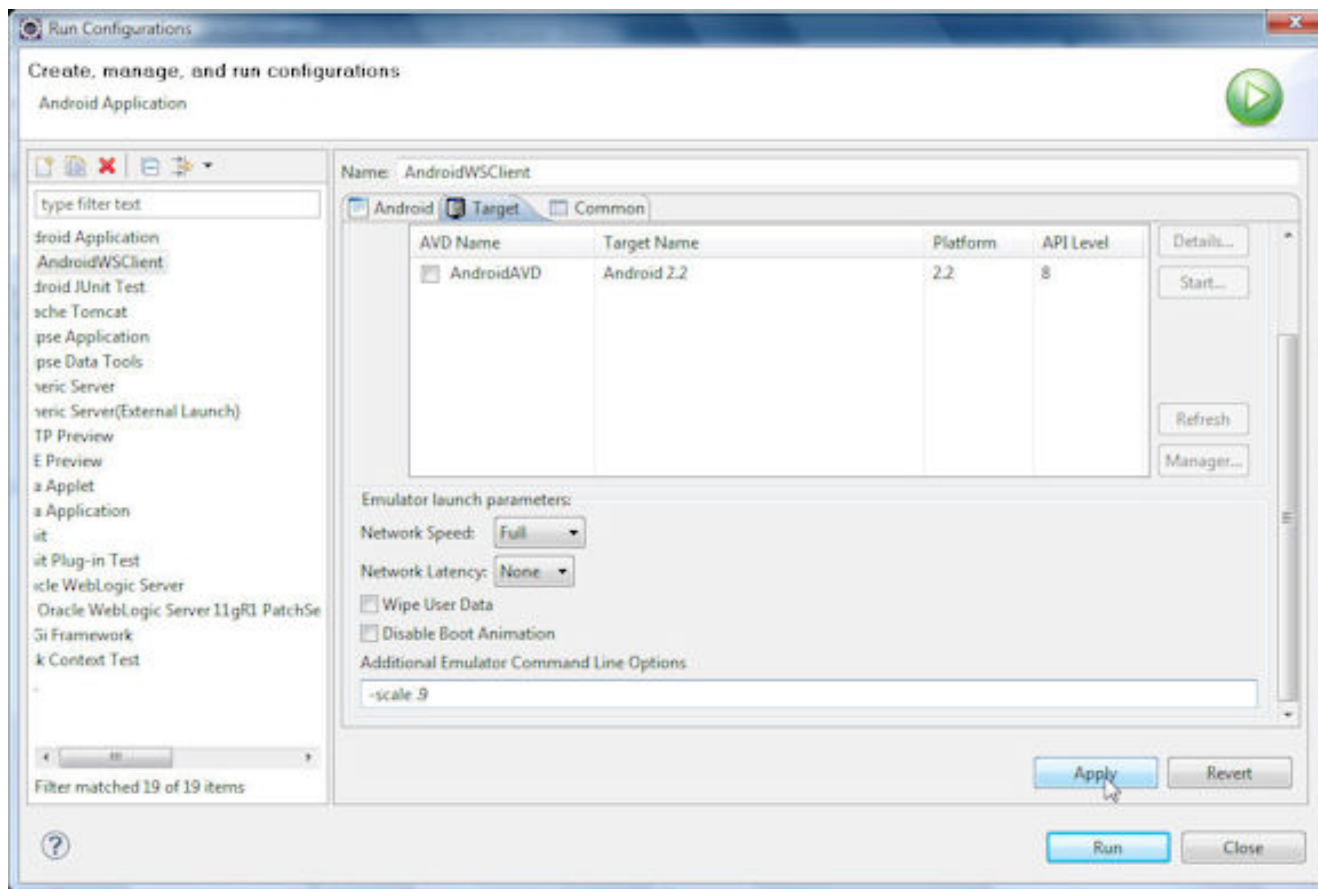
## Running the Android Client to access the web service

The Android emulator provides various options to customize the emulator. For example, to scale the emulator specify the `scale` value with the following option.

```
-scale <value>
```

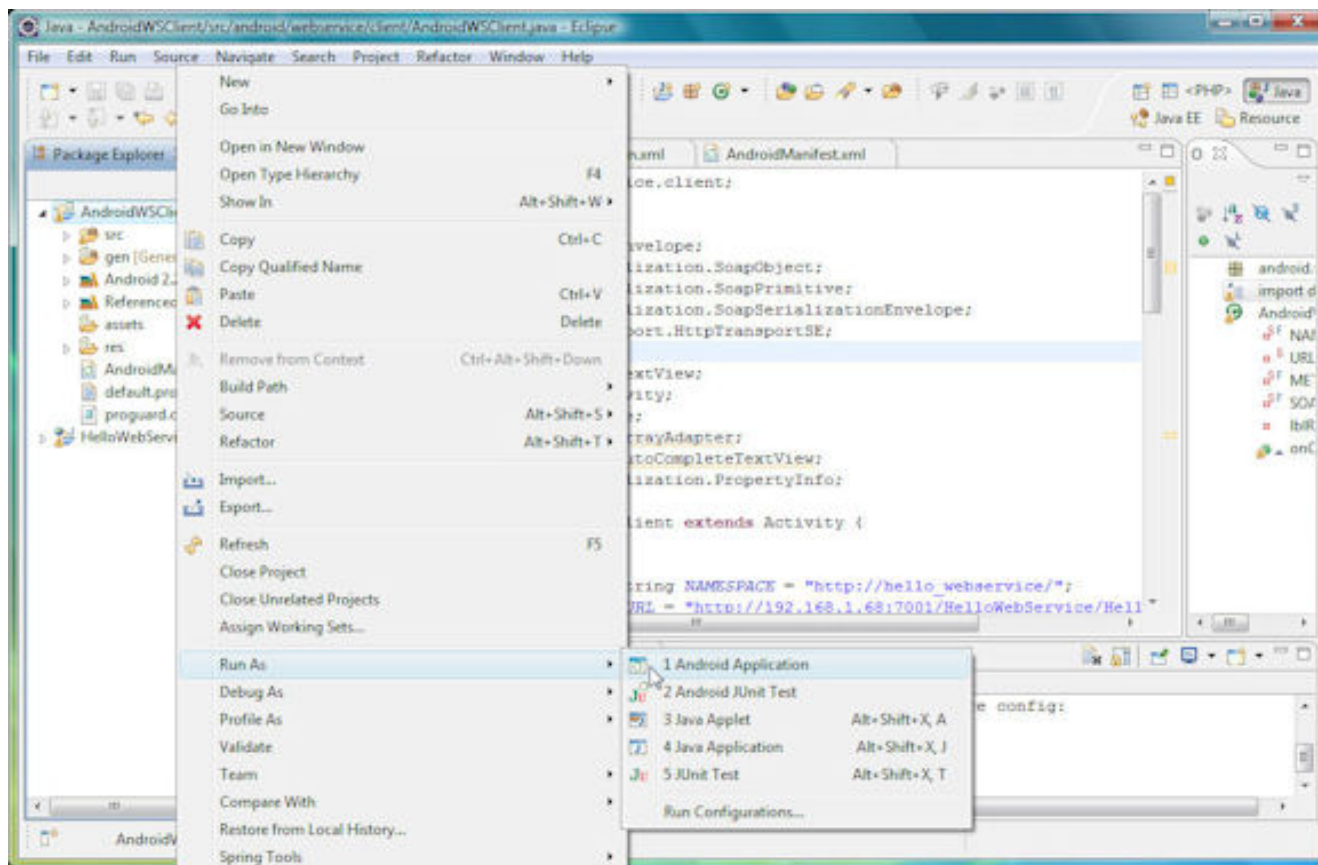
To specify an emulator option right-click on the `AndroidWSClient` application and select `Run As..>Run Configurations`. Select the `AndroidWSClient` application and specify the `-scale` option in field `Additional Emulator Command Line Options` and click on `Apply` as shown in Figure 17.

Figure 17. Setting the Emulator scale Option



To install and run the AndroidWSClient application on the emulator right-click on AndroidWSClient and select Run As...>Android Application as shown in Figure 18.

**Figure 18. Running the Android Application**



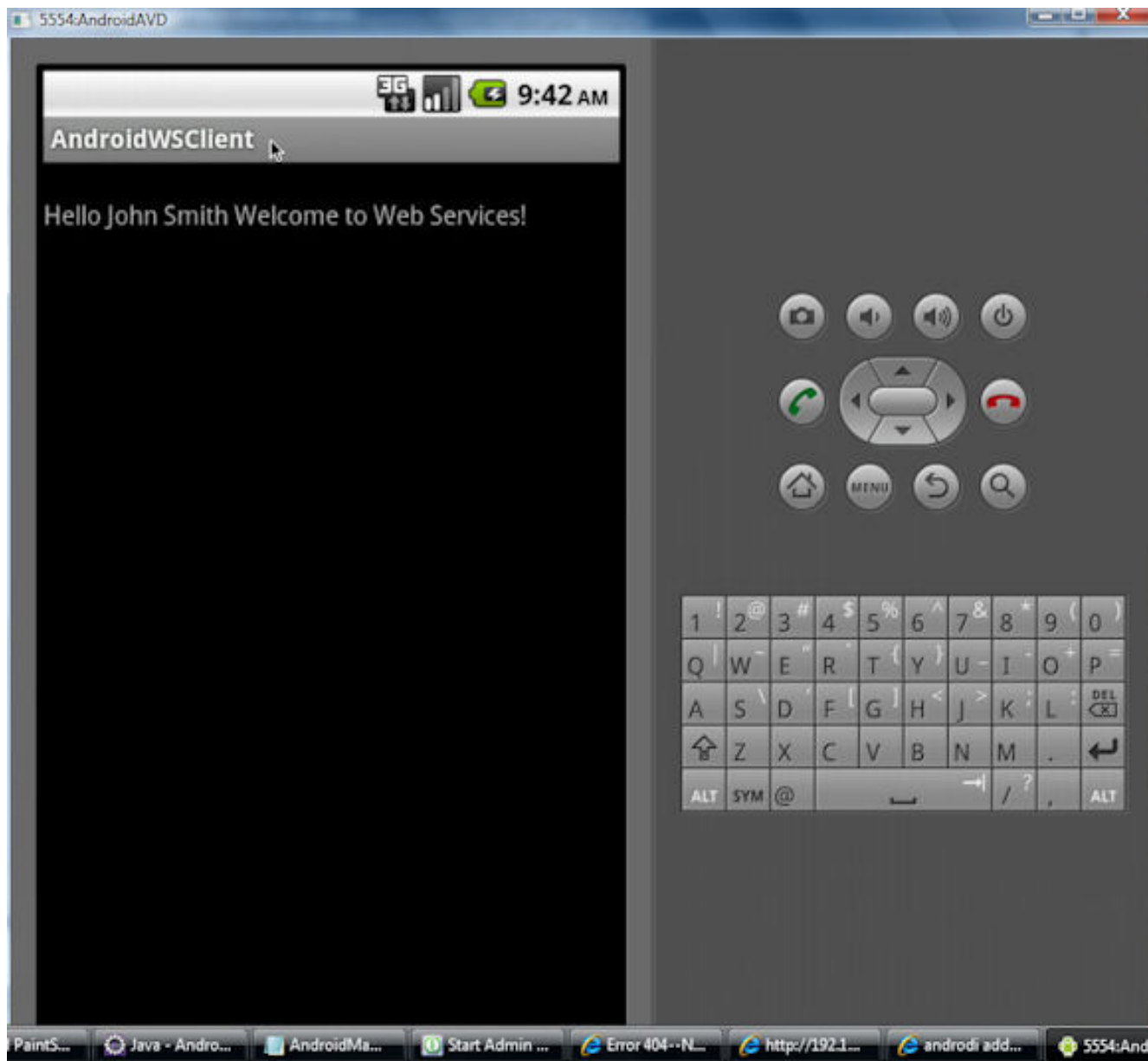
The AndroidWSClient application gets installed on the AVD as shown in Figure 19. Click on the application to run the web service client in the AVD.

**Figure 19. Android web service client installed**



The web service gets invoked and the response from the web service gets displayed in the Android emulator as shown in Figure 20.

**Figure 20. Accessing the web service**



The output from the running the Android application is shown in Listing 6.

#### Listing 6. Output from Android Application

```
AndroidWSClient] Android Launch!  
AndroidWSClient] adb is running normally.  
AndroidWSClient] Performing android.webservice.client.AndroidWSClient activity launch
```

```
AndroidWSClient] Automatic Target Mode: launching new emulator with compatible AVD
'AndroidAVD'
AndroidWSClient] Launching a new emulator with Virtual Device 'AndroidAVD'
AndroidWSClient] New emulator found: emulator-5554
AndroidWSClient] Waiting for HOME ('android.process.acore') to be launched...
AndroidWSClient] HOME is up on device 'emulator-5554'
AndroidWSClient] Uploading AndroidWSClient.apk onto device 'emulator-5554'
AndroidWSClient] Installing AndroidWSClient.apk...
AndroidWSClient] Success!
AndroidWSClient] Starting activity android.webservice.client.AndroidWSClient on device
emulator-5554
AndroidWSClient] ActivityManager: Starting: Intent { act=android.intent.action.MAIN
cat=[android.intent.category.LAUNCHER]
cmp=android.webservice.client/.AndroidWSClient }
```

When the Android application is run the following tasks are performed.

1. A new emulator gets launched
2. The `AndroidWSClient.apk` gets uploaded to the emulator
3. The `AndroidWSClient.apk` gets installed
4. The `android.webservice.client.AndroidWSClient` activity gets started

## Resources

### Learn

- Learn about [Android](#). Tools and documentation on how to create Android applications.
- In the [SOA and web services zone on developerWorks](#), get the project resources you need to advance your web services skills.
- ["Develop and deploy JAX-WS Web services on WebSphere Application Server Community Edition V2.0"](#) (developerWorks, Sept 2007) as a tutorial on developing and deploying JAX-WS web services with WebSphere application server.

### Get products and technologies

- [Download the Android SDK.](#)
- [Download the ksoap2-android library.](#)
- [Download the latest version of Eclipse for Java EE.](#)






## About the author

Deepak Vohra is a Sun Certified Java programmer, and Sun Certified Web Component Developer. Deepak has published in Java Developer's Journal and XML Journal.

## Rate this article

 Average rating (110 votes)

- ☐   1 star
- ☐   2 stars
- ☐   3 stars
- ☐   4 stars
- ☐  5 stars



## Comments

### Add comment:

[Sign in](#) or [register](#) to leave a comment.

Note: HTML elements are not supported within comments.

☐ Notify me when a comment is added

1000 characters left



Total comments (19)

Show:



I would like to recommend the use of Json Services, a web services framework that is lightweight and support Android proxy generation, it is hosted at <http://code.google.com/p/json-services/>

Posted by [Kumait](#) on 12 September 2012

[Report abuse](#)

Thanks for this tutorial,  
helped me call my webservice

Posted by [CJWN pavan kumar](#) on 30 April 2012

[Report abuse](#)

thanks!

Posted by [traduc](#) on 11 April 2012

[Report abuse](#)

NetworkOnMainThreadException is only thrown for applications with the Honeycomb SDK (Android 3.0) or higher.  
Use an earlier Android SDK version, or run the code in AsyncTask.  
<http://developer.android.com/reference/android/os/AsyncTask.html>

Posted by [dvohra09](#) on 29 February 2012

[Report abuse](#)

Is it not possible to utilize the wsdl artifacts you get from wsimport to create a client on the android ? Can Anyone explain why so?  
I'm getting NetworkOnMainThreadException when i tried this example, is there any solution for this...

Posted by [J7YU Amaan Khan](#) on 28 February 2012

[Report abuse](#)

[↑ Back to top](#)

[Print this page](#)

[Share this page](#) ▼

[Follow developerWorks](#) ▼

[About](#)

[Feeds and apps](#)

[Report abuse](#)

[Faculty](#)

[Help](#)

[Newsletters](#)

[Terms of use](#)

[Students](#)

[Contact us](#)

[IBM privacy](#)

[Business Partners](#)

[Submit content](#)

[IBM accessibility](#)