Wireless Vehicle Bus Adapter (WVA)

Android Library Tutorial

Revision history—90001431-13

Revision	Date	Description
A	October 2014	Original release.
B	October 2017	■ Rebranded the document. ■ Edited the document.

Trademarks and copyright

Disclaimers

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International. Digi provides this document "as is," without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

Warranty

To view product warranty information, go to the following website:

www.digi.com/howtobuy/terms

Send comments

Documentation feedback: To provide feedback on this document, send your comments to techcomm@digi.com.

Customer support

Digi Technical Support: Digi offers multiple technical support plans and service packages to help our customers get the most out of their Digi product. For information on Technical Support plans and pricing, contact us at +1 952.912.3444 or visit us at www.digi.com/support.

Contents

Step 4: Configure the WVA

Step 5: Subscribe to vehicle data on the WVA

Step 6: Create an alarm on vehicle data

WVA Android Library Tutorial

This tutorial introduces the Wireless Vehicle Bus Adapter (WVA) Android library and has examples of how to use the WVA Android library APIs to create an Android application for the WVA.

In the examples, you will learn how to read vehicle data using direct web-services and by configuring the event channel of the WVA. You can configure the event channel to asynchronously report events using the Subscription and Alarm WVA features. You will also learn how to programmatically access and change the configuration variables of the WVA and keep its time synchronized to your Android device.

What is the WVA library?

Digi has a Java-based library for Android devices that provides an API for accessing the web services of a WVA without needing to directly manage HTTP details. This library has been made open-source under the Mozilla Public Library (2.0) and is hosted on Github.

Options for using this tutorial and library

You can follow one of these methods to use this tutorial and the WVA Android library:

1. Follow the instructions from the beginning and develop the application with the aid of these instructions.
2. For the steps that interest you most, check out the branch for each step from the Git repository.
    - Each step in the tutorial has had a branch created that you can use as a base to check out and develop code.
    - If you follow this approach, make sure you follow the steps in the Getting Started section to confirm that you have the appropriate tools.
    - Using Git for source code version control is optional. You can simply follow the steps of this tutorial without cloning code from Git.

Getting started

Before you can use the the Wireless Vehicle Bus Adapter (WVA) Android library, you must install several tools and set up access to the tutorial. When setup is complete, you can build an Android project and work through the tutorial.

- Install tools and import the WVA tutorial.
- When you have completed the setup tasks, you are ready to begin the WVA Library Tutorial with Step 1: Build a shell Android Project with the WVA library.

Install tools and import the WVA tutorial

This section describes how to install the required tools, and set up and import the tutorial.

Step 1: Install Java and Git

Your computer will need the following tools installed or adjusted:

- A Java Development Kit (JDK) available here. The Java Android Studio issues a warning if a JDK is not installed.
- Git, available here.

Step 2: Install Android Studio

This tutorial uses Android Studio to develop the sample application. If you do not already have Android Studio installed, please go to the Android Developer site and follow the directions to install it. Also, ensure that you have installed at least one SDK matching the Android platform you wish to target.

You should install the latest version of Android Studio, even if that version is a beta release.

If you are new to Android Studio, you should review the Android build tools documentation.

Before you begin creating the application project and learning more about the WVA Library, verify that you can successfully run Android Studio without error.

Step 3: Get the WVA tutorial repository

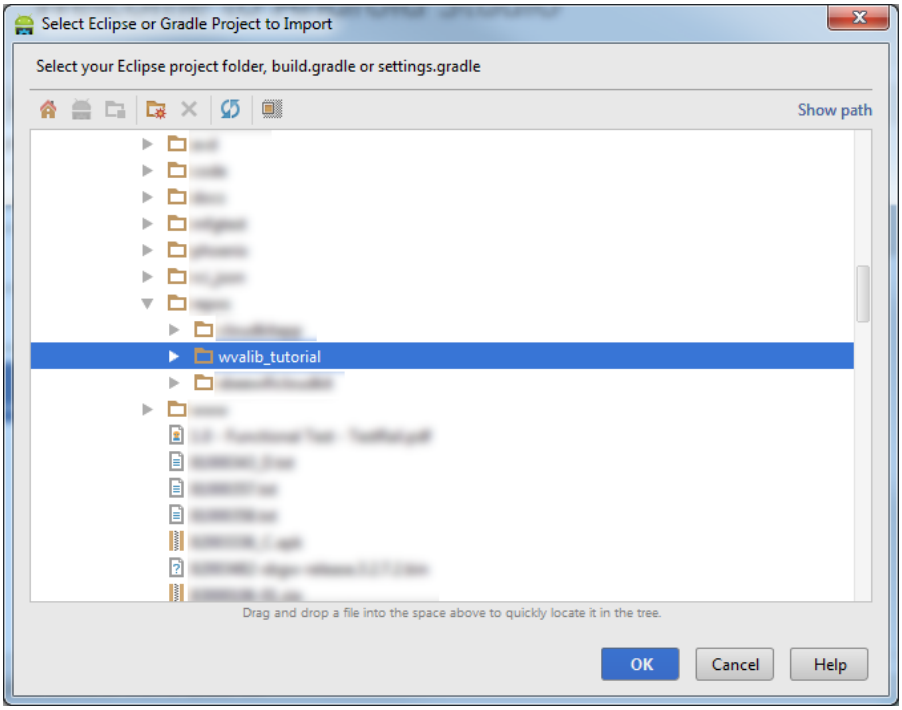
Using Git, clone the repository https://github.com/digidotcom/wvalib_tutorial. This is optional, but allows you to refer to each step compared to what you have created or to begin directly at a particular example.

Step 4: Import the WVA tutorial into Android Studio

To import the tutorial into Android Studio, use the **Import Project** option on either the quick-start page of the Android Studio or under the **File** menu, and select the cloned repository's root directory. By default this is **wvalib_tutorial**. Note that the name may be different if you cloned it to a different directory or have renamed it.

Depending on your device screen size, some text may be cut off. You can adjust the dimensions in the XML layout additions, specifically margin values, to fit the screen of the Android device being used.

The library configures the event channel of the WVA for JSON-formatted data. The format of the event channel follows the web services requests being made. Making requests for XML data independent of the library disrupts the library's operation.



Step 1: Build a shell Android Project with the WVA library

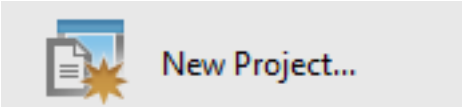
This section shows you how to create a new application using the WVA library. It assumes you have completed all the requirements of the Getting started section.

Follow the steps below to complete the process:

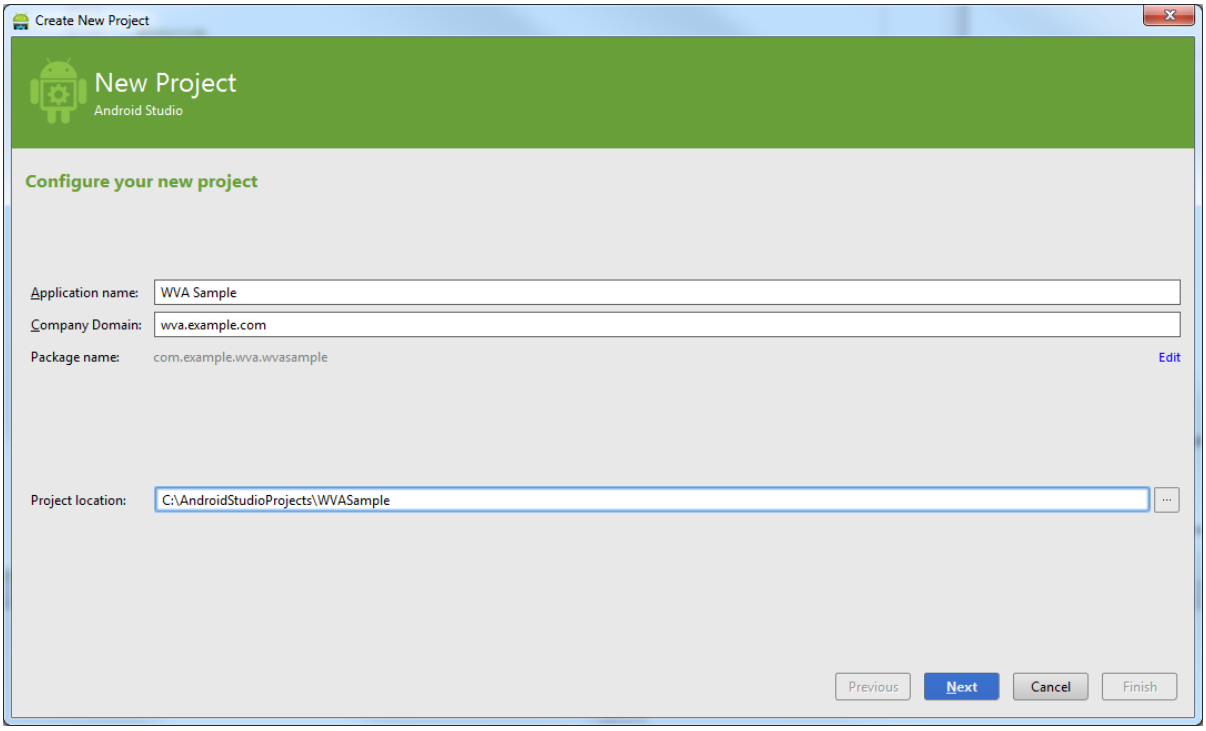
1. Create the Android project
2. Add the WVA library dependency
3. Create resources for use in your application
4. Add code to connect to the WVA
5. Build and run the application

Create the Android project

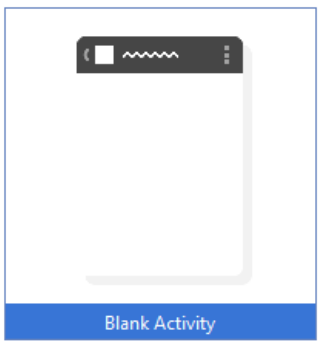
1. Open Android Studio.
2. From the quick start select **New Project**.



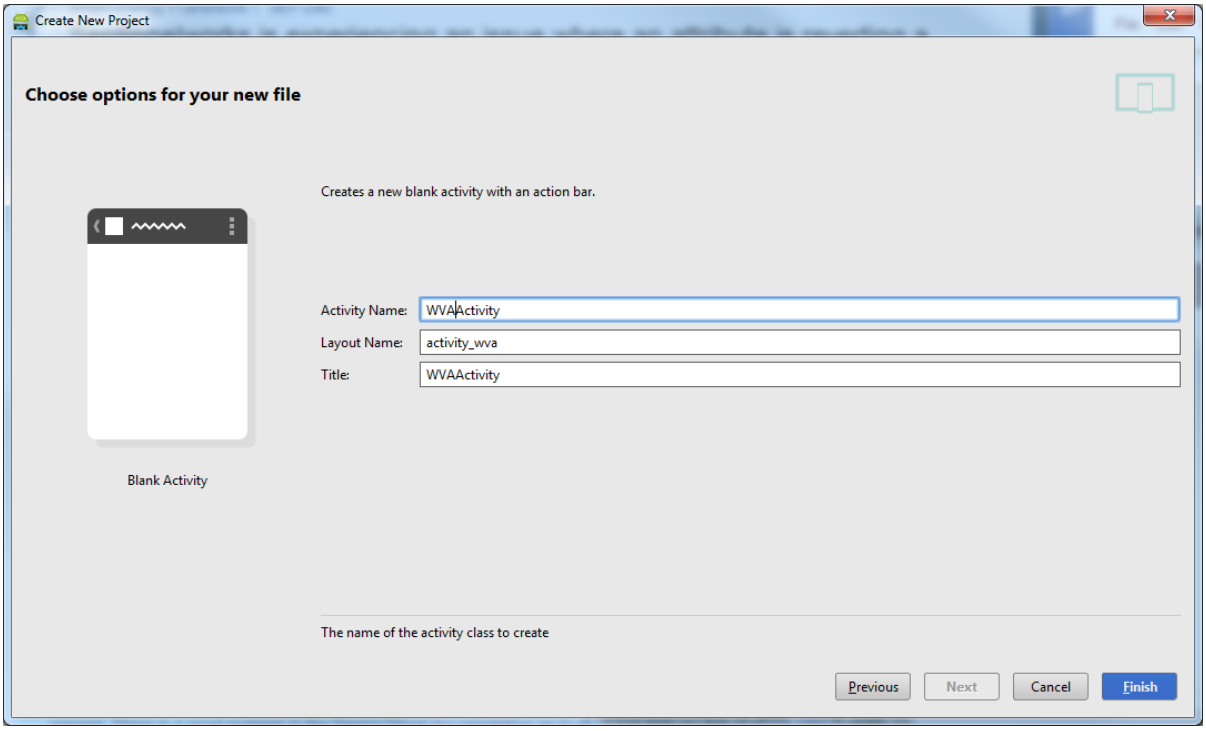
   a. For the **Application** name enter **WVA Sample**.
   b. For **Company Domain** enter **wva.example.com**.
   c. The **Project** location provides a default or you may change this as desired.
   d. Click **Next**.



3. On the **Select the form factors your app will run on** screen, leave the default option selected: **Phone and Tablet**.
4. Click **Next**. The **Add an activity to Mobile** screen appears.
5. Select **Blank Activity**.



6. Click **Next**. The **Choose options for your new file** screen appears.
7. Change **Activity Name:** to **WVAActivity**. This changes the **Layout Name** and **Title** to **activity_wva** and **WVAActivity**, respectively.

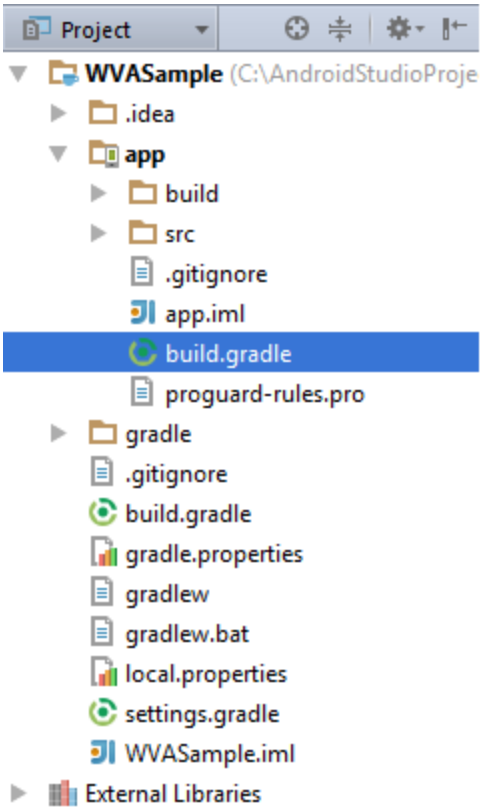


8. Click **Finish**.
9. Your project is now created. See Add the WVA library dependency for the next step in this process.

Add the WVA library dependency

This step explains how to add the WVA library from the JCenter central repository.

1. In your Project view tree, open **WVASample > app > build.gradle**:



2. You will need to synchronize your Gradle files in Android Studio (from the menu bar **Tools > Android > Sync Project with Gradle Files**) to pull the library from the Jcenter repository. In the dependencies section of this **build.gradle**, add the following line of code to add the WVA library to your project:

```
compile "com.digi.wva:wvalib:2.0+"
```

3. Create resources for use in your application.

Create resources for use in your application

This step explains how to create resources for use in your application.

1. Create some string resources to report the connection status to your WVA. In the Project View tree, navigate to and open **app > src > main > res > values > strings.xml,** and add the following string resource tags:

```
<string name="wva_connect_unknown">Waiting to connect.</string>
<string name="wva_connect_ok">WVA connection established!</string>
<string name="wva_connect_error">Cannot connect to WVA!</string>
```

2. Create a layout resource to display the text. In the Project View tree, navigate to and open **app > src > main > res > layout > activity_wva.xml**, remove the existing **TextView** resource from the layout, and then replace it with the following:

```
<TextView
  android:id="@+id/connection_status_text"
  android:text="@string/wva_connect_unknown"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" />
```

3. Next, Add code to connect to the WVA.

Add code to connect to the WVA

This step explains how to add the code needed to connect to the WVA. Since the WVA connection needs to persist beyond the Android activity life cycle, you need to put the code in an Application class.

Note You need to add the appropriate import statements for the symbols used in the code. The IDE can do this for you automatically. When you enter code, click on the incorrect symbol, press **<ALT>+<ENTER>**, and choose to allow the IDE to include the necessary imports. You can also do this for multiple symbols at once with the **Code>Optimize Imports...** menu item.

1. In the Project View tree, expand **WVASample>app>src>main>java>com.example.wva.wvasample**.
2. Right click **com.example.wva.wvasample** and select **New>Java Class** from the pop-up menu to add a new Java class.
3. Name the new class **WVAApplication**. This class needs to extend from the Android Application class in order to create a context for the application's life cycle for networking code.
4. Click **OK**. In the source view, place the cursor after the class and type:

```
extends Application
```

5. Define two private data members: **wva** of type **WVA** and **wva_ip** of type **String**.
6. Initialize the **wva_ip** string with the IP address of your Wireless Vehicle Bus Adapter. If you do not know your WVA's IP address, see Discover devices using ADDP for information on performing a device discovery.
7. Create a public getter method for the **wva** object by right clicking on the **wva** member that you just created, selecting **Generate...** and clicking the **Getter** option.
8. Override the method **onCreate**. You can right click on Application and select **Generate>Override Methods...** to create the skeleton body for **onCreate**.
9. In the **onCreate** method, add the code shown to initialize the WVA object. To configure the code, you can use basic authentication with the **useBasicAuth** method. Use the username and password of your device.

10. Configure the WVA object by using HTTPS with the **useSecureHttp** method. The following code shows what your class should look like after you are done:

```
public class WVAApplication extends Application {

    private String wva_ip = "w.x.y.z";
    private WVA wva;

    public WVA getWVA() {
        return wva;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        wva = new WVA(wva_ip);
        wva.useBasicAuth("admin","admin").useSecureHttp(true);
    }
}
```

11. Make sure your **AndroidManifest.xml** refers to this new application class, and give your app permission to access the networking code. In the Project view tree, navigate to **WVASample>app>src>main>res>AndroidManifest.xml** and add the following attribute to the **<application>** tag:

```
android:name=".WVAApplication"
```

12. Just above the **<application>** tag, add the following tag to enable networking permissions:

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

13. Add some code to the **WVAActivity** class you created earlier when you created the project. Open the class **WVAActivity** in the IDE, and add the following code to the **onCreate** method:
    a. Add the following variable declaration. This declaration creates a reference to our custom **WVAApplication** object:

```
final WVAApplication wvaapp = (WVAApplication)
getApplication();
```

b. Add a variable to reference the **TextView** to update the connection status in the display. Note that this refers to the resource **connection_status_text** you created earlier.

```
final TextView t = (TextView) findViewById(R.id.connection_
status_text);
```

c. At the end of **onCreate**, call the method **isWVA** to test if the WVA is connected to the same IP network as your Android device. The method expects a callback and so define it as an anonymous inline class by creating a callback object as follows. Note the callback object requires a single method **onResponse(Throwable, Boolean)** to be defined. This code adds the logic to display different text resources depending on the status of the connection to the WVA as passed in by the variable success.

```
wvaapp.getWVA().isWVA(new WvaCallback<Boolean>() {
    @Override
    public void onResponse(Throwable error, Boolean success)
{
        if(error != null) {
            error.printStackTrace();
        }
        else {
            if (success) {
                t.setText(R.string.wva_connect_ok);
            }
            else {
                t.setText(R.string.wva_connect_error);
            }
        }
    }
});
```

The code for the **onCreate** method should look like this example:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_wva);
    // Check that we are talking to a WVA
    final WVAApplication wvaapp = (WVAApplication) getApplication();

    final TextView t;

    t = (TextView) findViewById(R.id.connection_status_text);
```

```
    wvaapp.getWVA().isWVA(new WvaCallback<Boolean>() {
        @Override
        public void onResponse(Throwable error, Boolean success) {
            if (error != null) {
                error.printStackTrace();
            }
            else {
                if (success) {
                    t.setText(R.string.wva_connect_ok);
                }
                else {
                    t.setText(R.string.wva_connect_error);
                }
            }
        }
    });

}
```

14. See Build and run the application for the next step.

Discover devices using ADDP

You may need to use the Digi-proprietary protocol ADDP (Advanced Device Discovery Protocol) to discover your WVA to determine its IP address. Android applications can also use ADDP to discover WVA devices on your network. This section explains how to use Digi's ADDP library to perform discoveries.

Using ADDP from your computer

1. Using ADDP requires a Java JDK to be installed on your computer. Make sure your JDK was to your computer's PATH by opening a command prompt and entering the following:

   ```
   java -version
   ```

2. Download the ADDP library **.jar** file, using this direct link: download ADDP JAR file.
3. Optional: Rename the ADDP library **.jar** file to **AddpLibrary.jar** for convenience. While this step is optional, the rest of the steps in this procedure refer to the file by this name.
4. From a command prompt (Windows) or terminal (Linux/OS X), run the following command, which performs an ADDP discovery.

   ```
   java -jar AddpLibrary.jar
   ```

   Discovered devices are listed in a table, such as the one below:

   ```
         MAC Address |        IP Address |                      Hardware
   ========================================================================
   00:40:9D:5C:11:12 |       192.168.1.2 |  Wireless Vehicle Bus Adapter

   Search complete. 1 device(s) found.
   ```

5. For more verbose output, which displays all information for each device, add the **-v** argument:

```
java -jar AddpLibrary.jar -v
```

Using ADDP from an Android application

Digi's ADDP library is packaged for distribution via JCenter like the WVA Android library. To include the library as a dependency for your application, add the following line to the dependencies section of your **build.gradle** file:

```
compile "com.digi.addp:addplib:1.0"
```

To perform a discovery, first instantiate an **AddpClient** object, and then call **searchForDevicesAsync**. See the following code snippet:

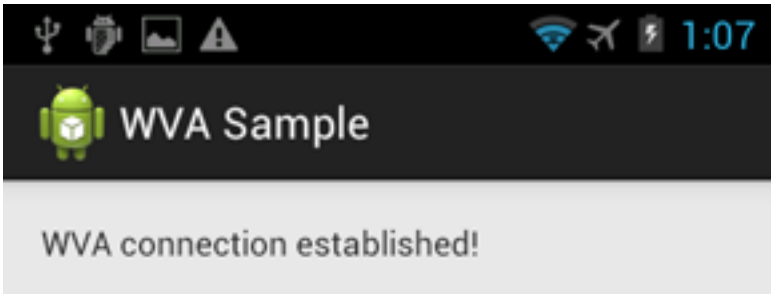
```
AddpClient client = new AddpClient();
client.searchForDevicesAsync(new DeviceFoundListener() {
    @Override
    public void onFound(String mac, AddpDevice device) {
        // Handle discovered device
    }

    @Override
    public void onSearchComplete() {
        // Handle completed search.
    }
});
```

Using this example, you can populate a **ListView** with a representation of each discovered device. If you plan to do this, be aware that this listener is invoked from a background thread. To change anything in your application's UI, you must pass the discovery information to the UI thread in some way.

For more information on this library's operations, see the Javadoc, available from the WVA Documentation page.

Build and run the application

Your application is now complete. Run it using the play button in the IDE. If properly configured on the same network as your WVA your application should look as follows:



Note If you receive an error related to the **getWVA** method in the build output, make sure the capitalization matches that of the generated method. For example, **getWva** versus **getWVA**.

Where can I find the completed source?

To check your work against the completed project, you can retrieve the entire project from the Git tutorial repository as specified in the Getting Started section and run the following command:

```
git checkout step1
```

Step 2: Synchronize time to the WVA

This section shows you how to add a time synchronization call to the simple application you have created so far. You can do so by adding a button to the activity created in Step 1: Build a shell Android Project with the WVA library, which uses the current time of your Android device to change the time of the WVA.

Your application needs to periodically synchronize the WVA clock.

Where do I begin?

Start with the project you created in Step 1: Build a shell Android Project with the WVA library, or begin here by performing the following command in the companion repository and using the project provided.

```
git checkout step1
```

Follow the steps below to complete the process:

1. Add UI elements
2. Create the button behavior
3. Build and run the application

Add UI elements

1. In the Project browser, navigate to and open **app > src > main > res > layout > activity_wva.xml**.
2. Add the following XML element to create the button that will trigger time synchronization:

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Set Time"
  android:id="@+id/set_time_button"
  android:layout_below="@+id/connection_status_text"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_marginTop="50dp" />
```

3. (Optional) If you would like a clock to be displayed within the application, add the following XML code:

```
<DigitalClock
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/digitalClock"
  android:layout_alignTop="@+id/set_time_button"
  android:layout_toRightOf="@+id/set_time_button"
  android:layout_toEndOf="@+id/set_time_button" />
```

4. See Create the button behavior for the next step.

Create the button behavior

1. Navigate to and open the code for the **WVAActivity** class you've been creating.
2. In the **onCreate** method, retrieve the button view created previously by adding:

```
final Button time_button = (Button) findViewById(R.id.set_time_button);
```

3. Define what you want to occur when the time button is clicked. At the top level of the **WVAActivity** class, create a new private void method called **time_button_clicked**, as follows. This function will look familiar, as all of the asynchronous calls in the WVA library follow the same form and **setTime** is no different. All that the library needs is a **Joda DateTime** object containing the time to set and a **WvaCallback** object for reporting status. A more complicated application would do something more useful with the success or failure of the call.

```
private void time_button_clicked(WVAApplication wvaapp) {
    WVA wva = wvaapp.getWVA();
    DateTime now = new DateTime();

    wva.setTime(now, new WvaCallback<DateTime>() {
        @Override
        public void onResponse(Throwable error, DateTime dateTime) {
            if (error != null) {
                error.printStackTrace();
            }
        }
    });
}
```

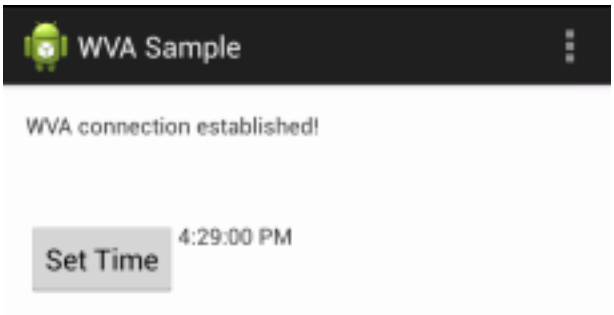
4. Arrange for the function just created to be called when the button is clicked by adding the following code at the end of **onCreate**.

```
time_button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        time_button_clicked(wvaapp);
    }
});
```

5. See Build and run the application for the next step.

Build and run the application

It may be necessary to clean up the imports of new symbols added during the steps in this process. However, once this has been done, the application should build and run on your device. When the **Set Time** button is pressed you can use the Web interface of the WVA to observe the time change. Try changing your Android time and observe as the time changes.



Where can I find the completed source?

If you would like to check your work against the completed project, you can retrieve the entire project from the Git tutorial repository as specified in the Getting Started section, and run the following command:

```
git checkout step2
```

Step 3: Read vehicle data from the WVA

This section shows you how to add a vehicle data query call to the simple application you have created so far. You can do so by adding a button to the activity created previously which fetches a vehicle data point from the WVA device.

Note If your application requires periodic updates of vehicle data, see Step 5: Subscribe to vehicle data on the WVA, which configures the WVA to provide data directly to your application rather than polling the WVA's web services.

Where do I begin?

You can start with the project you worked on in Step 2: Synchronize time to the WVA, or begin here by performing the following command in the companion repository and using the project provided.

```
git checkout step2
```

Follow the steps below to complete the process:

1. Add UI elements
2. Create button behavior
3. Build and run the application

Add UI elements

1. In the Project browser, navigate to and open **app > src > main > res > layout > activity_wva.xml**.
2. Add the following XML element to create the button that will trigger a vehicle data query:

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Fetch EngineSpeed"
  android:id="@+id/fetch_data_button"
  android:layout_below="@+id/set_time_button"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_marginTop="50dp" />
```

3. Add the following XML so that you can display the fetched value inside the application:

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/vehicle_data_value"
  android:layout_alignTop="@+id/fetch_data_button"
  android:layout_toRightOf="@+id/fetch_data_button"
  android:layout_toEndOf="@+id/fetch_data_button" />
```

4. See Create button behavior for the next step.

Create button behavior

1. Navigate to and open the code for the **WVAActivity** class you've been creating.
2. In the **onCreate** method, retrieve the button view created previously by adding:

```
final Button data_button = (Button) findViewById(R.id.fetch_data_button);
```

3. Define what should occur when this button is clicked. At the top level of the **WVAActivity** class create a new private void method called **data_button_clicked** as follows.

```
private void data_button_clicked(WVAApplication wvaapp) {
    WVA wva = wvaapp.getWVA();
    String endpoint = "EngineSpeed";

    final TextView value_view = (TextView) findViewById(R.id.vehicle_data_
value);

    wva.fetchVehicleData(endpoint, new WvaCallback<VehicleDataResponse>() {
        @Override
        public void onResponse(Throwable error, VehicleDataResponse
response) {
            if (error != null) {
                error.printStackTrace();
            } else {
                value_view.setText(Double.toString(response.getValue()));
            }
        }
    });
}
```

4. Define a **String** to indicate which vehicle data endpoint to query (in this case, **EngineSpeed**), and a **WvaCallback** object for reporting the response. To display the timestamp of the fetched

value, change the **setText** call to the following:

```
value_view.setText(String.format("%.3f\n%s",
                                 response.getValue(),
                                 response.getTime().toString()));
```

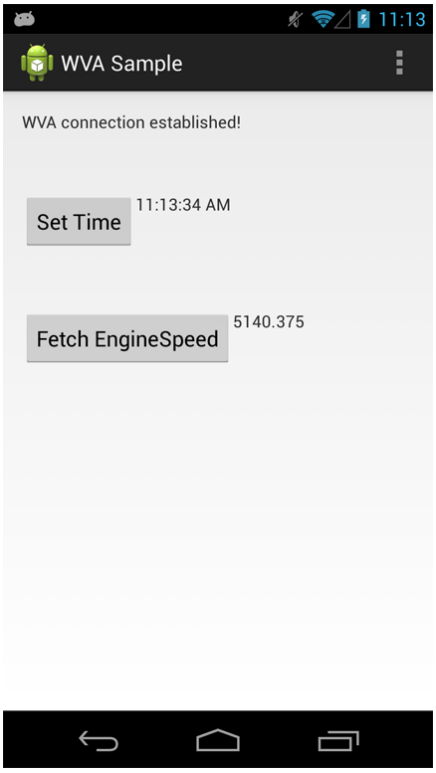
5. Add the following code at the end of **onCreate**. This calls the previously created function when the button is clicked.

```
data_button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        data_button_clicked(wvaapp);
    }
});
```

6. See Build and run the application for the next step.

Build and run the application

It may be necessary to clean up the imports of new symbols added during this process. You can use Android Studio's **Optimize imports** function to do this (from the menu bar **Code > Optimize Imports...** or type **CTRL+ALT+O**). Once this has been done, the application should build and run on your device. When the **Fetch EngineSpeed** button is pressed, you should see that value appear on the screen.



Where can I find the completed source?

To check your work against the completed project, retrieve the entire project from the Git tutorial repository as specified in the Getting started section and run the following command:

```
git checkout step3
```

Step 4: Configure the WVA

This section shows how to add a configuration call to the simple application you have created so far. You will add two buttons to the activity created previously, which enable and disable the HTTP web server, respectively.

Where do I begin?

Start with the project you worked on in Step 3: Read vehicle data from the WVA, or enter the following command in the companion repository and use the project provided.

```
git checkout step3
```

Follow the steps below to complete the process:

1. Add UI elements
2. Create button behavior
3. Build and run the application

Add UI elements

1. Navigate to and open **app > src > main > res > layout > activity_wva.xml** in the Project browser.
2. Add the following XML element to create the button that will enable the WVA's HTTP web server.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Enable HTTP"
  android:id="@+id/enable_http_button"
  android:layout_below="@+id/fetch_data_button"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_marginTop="50dp" />
```

3. Add the following XML element to create the button that will disable the WVA's HTTP web server.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Disable HTTP"
  android:id="@+id/disable_http_button"
  android:layout_alignTop="@+id/enable_http_button"
  android:layout_toRightOf="@+id/enable_http_button"
  android:layout_toEndOf="@+id/enable_http_button" />
```

4. See Create button behavior for the next step.

Create button behavior

1. Navigate to and open the code for the **WVAActivity** class you've been creating.
2. In the **onCreate** method, retrieve the button views created previously by adding the following:

```
final Button enable_http_button = (Button) findViewById(R.id.enable_http_
button);
final Button disable_http_button = (Button) findViewById(R.id.disable_http_
button);
```

3. Define what should occur when these buttons are clicked. At the top level of the **WVAActivity** class, create a new private void method called **enable_http_clicked** as follows:

```
private void enable_http_clicked(WVAApplication wvaapp) {
    WVA wva = wvaapp.getWVA();

    JSONObject json = new JSONObject();
    try {
        json.put("enable", "on");
        json.put("port", 80);
    } catch (JSONException e) {
        // Unexpected error
        e.printStackTrace();
        return;
    }

    wva.configure("http", json, new WvaCallback<Void>() {
        public void onResponse(Throwable error, Void response) {
            if (error != null) {
                error.printStackTrace();
            } else {
                Toast.makeText(getApplicationContext(), "Enabled HTTP
server", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

The **wva.configure** method call takes a configuration web services path, a JSON object to be sent, and a **WvaCallback** for reporting the success or failure of the call. In this case, you are configuring the HTTP web server, and so you pass in**http"**. This translates into a request on the web service endpoint **/ws/config/http**.

If you are familiar with the WVA's web services, you might notice that the JSON object you created is not the complete JSON object that you need to send to the WVA. What you passed to the configure call still needs to be wrapped in another JSON object, as the value for the **http** key. To make the library easier to use, the configure method wraps the given JSON object with the path provided in the first argument. This will work with nearly any path, for example: **http**, **canbus/1config/interface/wlan0**, **/ws/config/http**, **http://192.168.100.1/ws/config/idigi**. The library does the correct thing with all of these paths.

In addition, create a new **disable_http_clicked** method as follows:

```
private void disable_http_clicked(WVAApplication wvaapp) {
    WVA wva = wvaapp.getWVA();

    JSONObject json = new JSONObject();
    try {
        json.put("enable", "off");
        json.put("port", 80);
    } catch (JSONException e) {
        // Unexpected error
        e.printStackTrace();
        return;
    }

    wva.configure("http", json, new WvaCallback<Void>() {
        @Override
        public void onResponse(Throwable error, Void response) {
            if (error != null) {
                error.printStackTrace();
            } else {
                Toast.makeText(getApplicationContext(), "Disabled HTTP
server", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

4. Add the following code at the end of **onCreate** to call the newly created functions when either button is clicked. You can use a single **OnClickListener** which calls the correct function based on the button's ID.

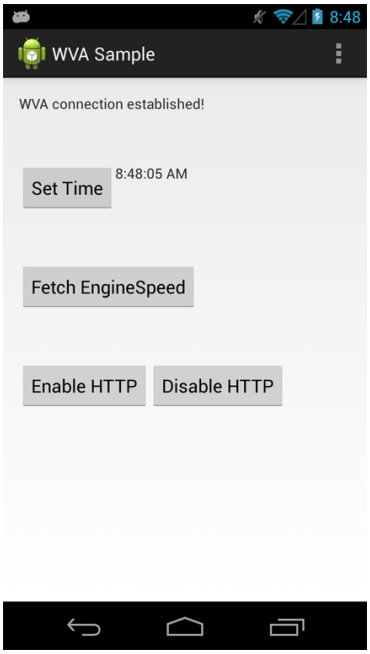
```
View.OnClickListener httpClick = new View.OnClickListener() {
@Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.enable_http_button:
                enable_http_clicked(wvaapp);
                break;
            case R.id.disable_http_button:
                disable_http_clicked(wvaapp);
                break;
        }
    }
};

enable_http_button.setOnClickListener(httpClick);
disable_http_button.setOnClickListener(httpClick);
```

5. See Build and run the application for the next step.

Build and run the application

You might need to clean up the imports of new symbols added during this process. To do this, use Android Studio's **Optimize imports** function (from the menu bar **Code > Optimize Imports...**, or type **CTRL+ALT+O**).

Once you have cleaned up the imports, build and run the application on your device. When you press the **Enable HTTP** button, you should be able to use the web interface of the WVA to observe that the HTTP web server has been enabled. Try navigating to **http://<WVA IP>**, or check the **Network Services** configuration. When you press the **Disable HTTP**, you should similarly be able to observe that the HTTP web server is disabled.



Note There is no automatic mechanism to switch use of HTTP or HTTPS in applications based on changes in the WVA's configuration settings. Make sure your application's use of HTTP/HTTPS matches the WVA's configuration settings for HTTP and HTTPS as network services. In the WVA's web interface, go to **Configuration > Network Services** and check the following settings: **Enable Web Server (HTTP)** and **Enable Secure Web Server (HTTPS)**, including the port numbers assigned for each service.

The following are indicators that there is a mismatch between the application code and the WVA's Network Services settings for HTTP/HTTPS:

- If HTTP is disabled and you attempt to use it
  - A redirect from HTTP to HTTPS occurs; that is, entering **http://** is redirected to **https://**.
  - On certain library calls, an error is issued with the message **Unexpected response body**. This error is caused by the redirect from **http://** to **https://**
- If HTTPS is disabled and you attempt to use it, a connection-error or **ECONNREFUSED** message is issued.

If both HTTP and HTTPS are disabled you can enable them in two ways:

- Use Device Cloud to reconfigure HTTP and HTTPS settings. From the Device Management device list, double-click the device to display the device properties menu. The settings are **Web user interface (HTTP)** and **Web user interface, secure (HTTPS)**.
- Reset the device to factory defaults by pressing and holding the WVA's button for ten seconds.

Where can I find the completed source?

To check your work against the completed project, you can retrieve the entire project from the Git tutorial repository as specified in the Getting Started section, and then run the following command:

```
git checkout step4
```

Step 5: Subscribe to vehicle data on the WVA

This section shows you how to use the WVA Android library to subscribe to vehicle data from the WVA. You can update the application to automatically connect to the WVA's event channel, add a button that sends the subscription request, and learn how to automatically receive new values coming in from the WVA device.

Where do I begin?

You can start with the project you worked on in Step 4: Configure the WVA, or begin here by performing the following command in the companion repository and using the project provided.

```
git checkout step4
```

Follow the steps below to complete the process:

1. Add UI elements
2. Create button behavior
3. Build and run the application

Add UI elements

1. Navigate to and open **app > src > main > res > layout > activity_wva.xml** in the Project browser.
2. Add the following XML element to create a button that creates a subscription to vehicle data.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Subscribe"
  android:id="@+id/subscribe_button"
  android:layout_below="@+id/enable_http_button"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_marginTop="50dp" />
```

3. Add the following XML so that you can display the newest value inside the application.

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/engine_speed_value"
  android:layout_alignTop="@+id/subscribe_button"
  android:layout_toRightOf="@+id/subscribe_button"
  android:layout_toEndOf="@+id/subscribe_button" />
```

4. See Create button behavior for the next step.

Create button behavior

1. Navigate to and open the code for the **WVAApplication** class you've been creating.
2. In the **onCreate** method, set up a new **EventChannelStateListener** and pass it to the **WVA** object. This listener is called to report various states of the event channel connection. This code makes it so that a toast notification appears on screen whenever the connection state changes.

```
wva.setEventChannelStateListener(new EventChannelStateListener() {
    @Override
    public void onConnected(WVA device) {
        Toast.makeText(getApplicationContext(), "Event channel connected!",
Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onError(WVA device, IOException error) {
        error.printStackTrace();
        Toast.makeText(getApplicationContext(), "Event channel error: " +
error.getMessage(), Toast.LENGTH_LONG).show();
            // If there is an unexpected error, disconnect from the event
channel.
        device.disconnectEventChannel(true);
    }

    @Override
    public void onRemoteClose(WVA device, int port) {
        Toast.makeText(getApplicationContext(), "Event channel closed on
remote side. Reconnecting...", Toast.LENGTH_SHORT).show();

        // Automatically reconnect after 15 seconds.
        this.reconnectAfter(device, 15000, port);
    }

    @Override
    public void onFailedConnection(WVA device, int port) {
        Toast.makeText(getApplicationContext(), "Couldn't connect event
channel", Toast.LENGTH_SHORT).show();
    }
});
```

3. In the **onCreate** method, below the code just added, call the WVA object's **connectEventChannel** method as follows. This call directs the library to open a TCP connection with the WVA, on port 5000.

```
wva.connectEventChannel(5000);
```

4. Navigate to and open the code for the **WVAActivity** class you have been creating.
5. In the **onCreate** method, retrieve the previously created views by adding:

```
final Button subscribe_button = (Button) findViewById(R.id.subscribe_
button);
final TextView engine_speed_value = (TextView) findViewById(R.id.engine_
speed_value);
```

6. In the **onCreate** method, arrange for the displayed value to be updated each time a new value arrives via the event channel. Do this by setting up a **VehicleDataListener**:

```
wvaapp.getWVA().setVehicleDataListener("EngineSpeed", new
VehicleDataListener() {
    @Override
    public void onEvent(VehicleDataEvent event) {
        VehicleDataResponse response = event.getResponse();
        engine_speed_value.setText(String.format("EngineSpeed = %.3f\n%s",
response.getValue(), response.getTime().toString()));
    }
});
```

7. At the top level of the WVAActivity class, create a new private void method called **subscribe_to_engine_speed** as follows:

```
private void subscribe_to_engine_speed(WVAApplication wvaapp) {
    WVA wva = wvaapp.getWVA();

    wva.subscribeToVehicleData("EngineSpeed", 15, new WvaCallback<Void>() {
        @Override
        public void onResponse(Throwable error, Void response) {
            if (error != null) {
                error.printStackTrace();
            } else {
                Toast.makeText(getApplicationContext(), "Subscribed to
EngineSpeed", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

8. Add the following code at the end of **onCreate** to call the function you just created when the **Subscribe** button is clicked.

```
subscribe_button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        subscribe_to_engine_speed(wvaapp);
    }
});
```

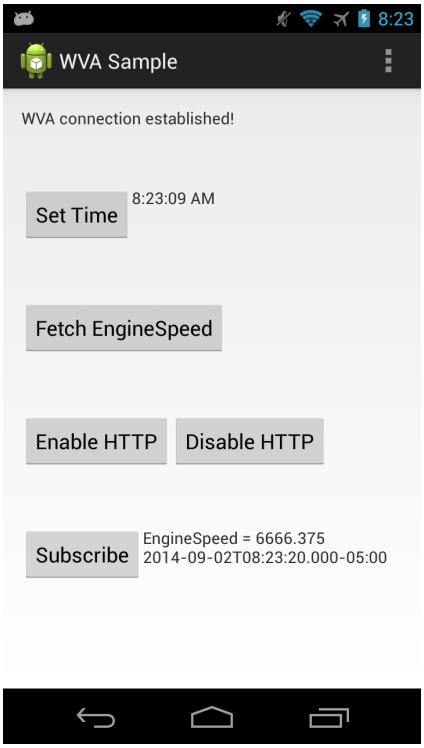
9. See Build and run the application for the next step.

Build and run the application

It may be necessary to clean up the imports of new symbols added during this process. You can use Android Studio's **Optimize imports** function to do this (from the menu bar **Code > Optimize Imports...** or type **CTRL+ALT+O**).

Once this has been done, the application should build and run on your device. When the **Subscribe** button is pressed you should start seeing new values for **EngineSpeed** vehicle data arrive to your application every 15 seconds.

Note If you have previously set up an **EngineSpeed** subscription on your WVA, then you automatically see data in the application without needing to press the **Subscribe** button.



You may wish to override the **onStop** or **onDestroy** method in **WVAActivity** to remove the **EngineSpeed** data listener. This is particularly important in cases where there are references that may keep objects from being garbage collected, such as the Activity being stopped. To remove the listener, call **removeVehicleDataListener** as follows:

```
@Override
protected void onDestroy() {
    super.onDestroy();

    WVAApplication app = (WVAApplication) getApplication();
    app.getWVA().removeVehicleDataListener("EngineSpeed");
}
```

Where can I find the completed source?

To check your work against the completed project, retrieve the entire project from the Git tutorial repository as specified in Getting started and run the following command:

```
git checkout step5
```

Step 6: Create an alarm on vehicle data

This section shows how to use the WVA Android library to create a vehicle data alarm on the WVA. You will update the application to automatically create a vehicle data alarm record, add a text view to display alarm values, and learn how to distinguish subscription and alarm events.

Where do I begin?

Start with the project you worked on in Step 5: Subscribe to vehicle data on the WVA, or enter the following command in the companion repository and use the project provided.

```
git checkout step5
```

Follow the steps below to complete the process:

1. Add UI elements
2. Create the alarm and handle alarm data
3. Build and run the application

Add UI elements

1. In the **Project** browser, navigate to and open **app > src > main > res > layout > activity_wva.xml**.
2. Add the following XML element to create the text view for displaying the alarm values:

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/engine_speed_alarm_value"
  android:layout_below="@+id/subscribe_button"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_marginTop="50dp" />
```

3. See Create the alarm and handle alarm data for the next step.

Create the alarm and handle alarm data

1. Navigate to and open the code for the **WVAActivity** class you've been creating. In the **onCreate** method, retrieve the text view created previously by adding:

```
final TextView engine_speed_alarm_value = (TextView) findViewById
(R.id.engine_speed_alarm_value);
```

In the **onCreate** method, arrange for the displayed value to be updated each time a new value arrives via the event channel. Do this by editing the **VehicleDataListener** created in Step 5: Subscribe to vehicle data on the WVA. Here you can see that you may use the **getType** method of an event to determine if a newly arrived event indicates new subscription or alarm data. Note that since this listener needs access to the **engine_speed_alarm_value** variable, you should place the declaration for **engine_speed_alarm_value** next to **engine_speed_value**.

```
wvaapp.getWVA().setVehicleDataListener("EngineSpeed", new
VehicleDataListener() {
    @Override
    public void onEvent(VehicleDataEvent event) {
        VehicleDataResponse response = event.getResponse();

        if (event.getType() == EventFactory.Type.SUBSCRIPTION) {
            engine_speed_value.setText(
                String.format("EngineSpeed = %.3f\n%s",
                              response.getValue(),
                              response.getTime().toString()));
        }
        else if (event.getType() == EventFactory.Type.ALARM) {
            engine_speed_alarm_value.setText(
            String.format("EngineSpeed (Alarm) = %.3f\n%s",
                          response.getValue(),
                          response.getTime().toString()));
        }
    }
});
```

2. Add the following code at the end of **onCreate** to create an alarm on **EngineSpeed** values. This code creates a new data alarm record which will generate an alarm event on the TCP event channel when **EngineSpeed** is above **1000**. The **10** argument directs the WVA to report this alarm condition at most once every ten seconds.

```
wvaapp.getWVA().createVehicleDataAlarm("EngineSpeed", AlarmType.ABOVE,
1000, 10, new WvaCallback<Void>() {
    @Override
    public void onResponse(Throwable error, Void response) {
        if (error != null) {
            error.printStackTrace();
        } else {
            Toast.makeText(getApplicationContext(),
                           "Created alarm on EngineSpeed",
                           Toast.LENGTH_SHORT)
                 .show();
        }
    }
});
```

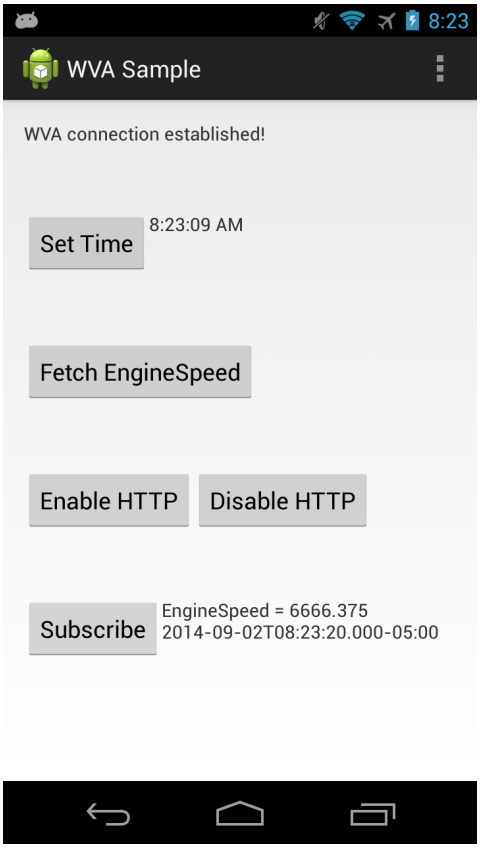
3. See Build and run the application for the next step.

Build and run the application

It may be necessary to clean up the imports of new symbols added during this process. You can use Android Studio's **Optimize imports** function to do this (from the menu bar **Code > Optimize Imports...** or type **CTRL+ALT+O**).

Once you have cleaned up the imports, the application should build and run on your device. When the application starts up, you should see a toast notification indicating that the alarm has been created. Next, you should start seeing new **EngineSpeed** alarm values appear, below the **Subscribe** button.

You will only see new values if the **EngineSpeed** value detected by your WVA is above **1000**. Therefore, you may need to adjust your simulator (if you are using one), or change the threshold value passed into the **createVehicleDataAlarm** call before you will see data.



Where can I find the completed source?

To check your work against the completed project, retrieve the entire project from the Git tutorial repository as specified in the Getting Started section, and then run the following command:

```
git checkout step6
```