# isCOBOL Evolve: Mobile for Android

**Key Topics:**



# Table of Contents

# Chapter 1

# Introduction

isCOBOL Mobile allows to bring COBOL code on mobile devices. The goal is to reuse the existing backend COBOL logic as well as sequential, relative and indexed files (in JIsam format) on a mobile application while the UI is rewritten using a HTML5/CSS3 UI Framework such as jQuery Mobile or Dojo Mobile or Sencha Touch.

The code described in this book takes advantage of JQuery Mobile for the UI. You can find documentation about JQuery Mobile at http://view.jquerymobile.com/1.3.0/.

## Technical Notes About isCOBOL Mobile

The COBOL program has to be transormed in a WebService REST stateful. This objective is achieved through a new internal class that allows to communicate with HTML pages through AJAX retrieving data and printing results.

The UI is rendered on mobile devices using the WebView component available on Android ADT while the COBOL logic runs locally on the device thanks to the new Framework library provided along with isCOBOL Mobile. Such library is compatible for Android version 3.0 or greater and also allows to interact with API functions of your mobile devices such as Phone Book, Memo, etc.

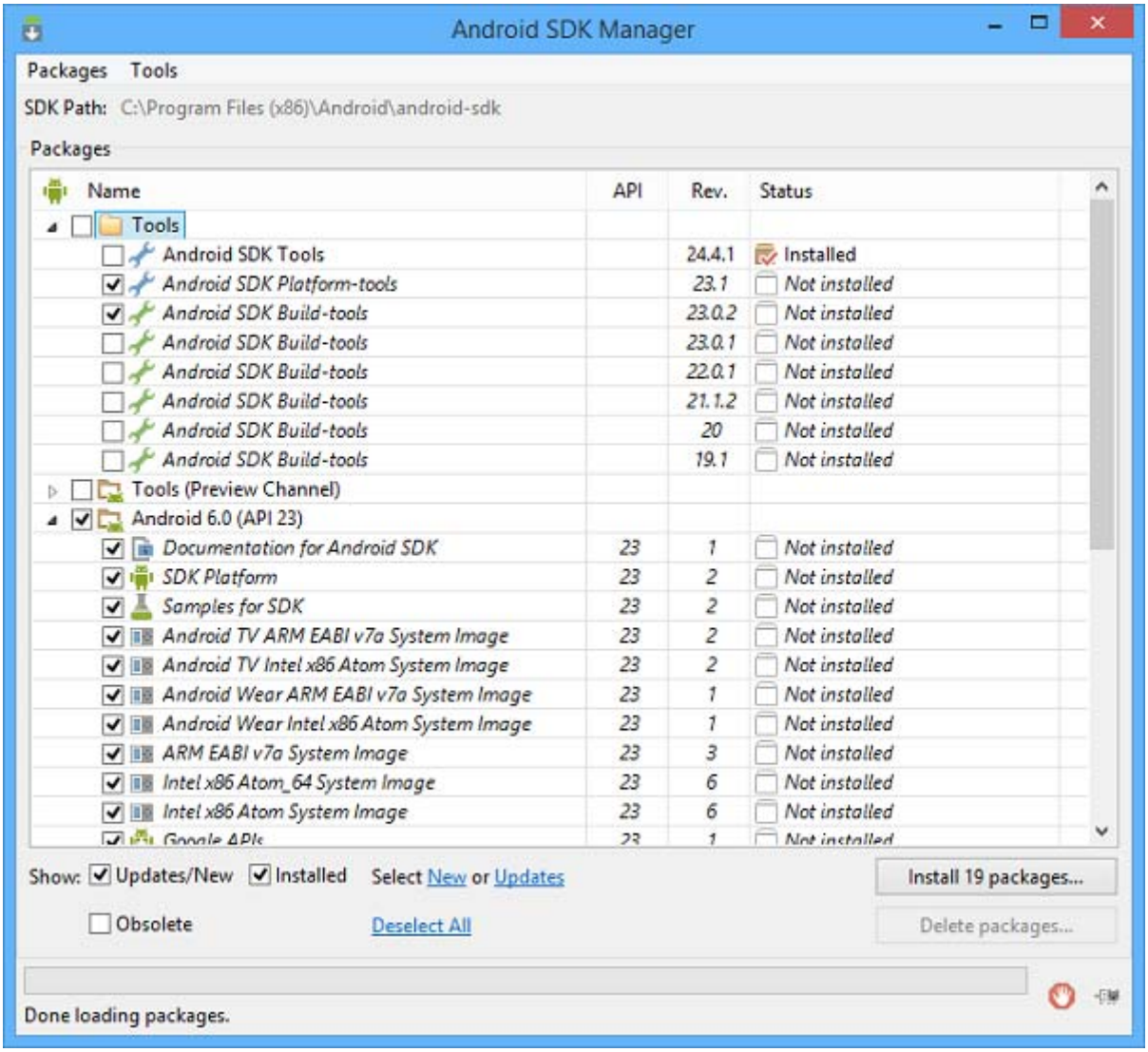
# Chapter 2

# Installation

## Environment

In order to deploy and run a mobile app, the following environment must be set up:

- Android SDK
  The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.
  The below instructions explain how to download and install the SDK on Windows. For information on how to install the SDK on other platforms, refer to the official site http://developer.android.com.
  In order to download the Android SDK, navigate to http://developer.android.com/sdk/index.html and scroll down to "Get just the command line tools". Choose the proper zip file according to the platform where you're going to develop. Aaccept the terms and conditions, then proceed with the download.
  Once the download is complete, unzip the archive in a folder of your choice, for example on Windows you could unzip it under *C:\AndroidSDK* and you obtain *C:\AndroidSDK\tools*. Run the android command (*android.bat*) within the *tools* folder in order to start the SDK manager.
  The SDK Manager is launched and it allows to download the necessary resources for development:




By default the SDK Manager allows to install the latest platform tools, system image and USB driver, but you can install also previous versions as well as additional resources. Mark the items that you wish to install, then click on the *Install packages...* button.

At the end of the installation, you will find a new folder named *platform-tools* at the same level of *tools*.

Once the Android SDK has been installed, add the *tools* and *platform-tools* directories of Android's SDK to the PATH environment variable in order to be able to run android commands from the command line.

If you wish to run the SDK Manager again, later:

- You can use the android command mentioned above.

- Alternatively, you can open a command prompt and execute:

```
android sdk
```

- Apache ANT (if you plan to work without isCOBOL IDE)
  Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The Android SDK takes advantage of ANT in order to generate the apk of your mobile app.

  The Apache ANT main page is http://ant.apache.org
  1. Download the latest binary files from http://ant.apache.org/bindownload.cgi
  2. Unzip the binary distribution in a folder of your choice (e.g. C:\ANT)
  3. Add the *bin* directory of ANT to the PATH environment variable

- (optional) Tomcat or another servlet container
  The servlet container allows to run your app as a web application making it usable by every device equipped with a web-browser including Apple iOS devices.

  The Apache Tomcat main page is http://tomcat.apache.org/
  Here are some example steps to download and install Tomcat 7 on Windows:
  **NOTE -** To avoid problems, uninstall earlier versions of the Tomcat service before installing Tomcat 7
  1. Make sure that you already have installed JRE 5 or 6
  2. Visit http://tomcat.apache.org/
  3. Click on the Tomcat 7.x Download link (on the left side)
  4. Find the Binary Distributions section and click on the Windows Service Installer link
  5. Run the downloaded executable file and follow the prompts accepting the defaults

- The isCOBOL Mobile Framework

```
ismobile.jar
```

# Activate the License

If you provided license keys during the installation, on Windows, you should skip reading this chapter.

isCOBOL Mobile looks for the following configuration properties for the license keys at compile-time:

```
iscobol.compiler.license.2018=<license_key>
iscobol.eis.license.2018=<license_key>
iscobol.mobile.license.2018=<license_key>
```

These keys should be stored in one of the following files (if they exist):

## Windows

1. \etc\iscobol.properties in the drive where the working directory is
2. C:\Users\<username>\iscobol.properties (the setup wizard saves licenses here, if you don't skip activation)
3. iscobol.properties found in the Java Classpath
4. %ISCOBOL%\iscobol.properties
5. a custom configuration file passed on the command line

### Unix/Linux

1. /etc/iscobol.properties
2. $HOME/iscobol.properties
3. iscobol.properties found in the Java Classpath
4. $ISCOBOL/iscobol.properties
5. a custom configuration file passed on the command line

**NOTE** - Files are listed in the order they're processed. If the license keys appears in more than one of the above files, then the last occurrence is considered.

## The AVD Manager

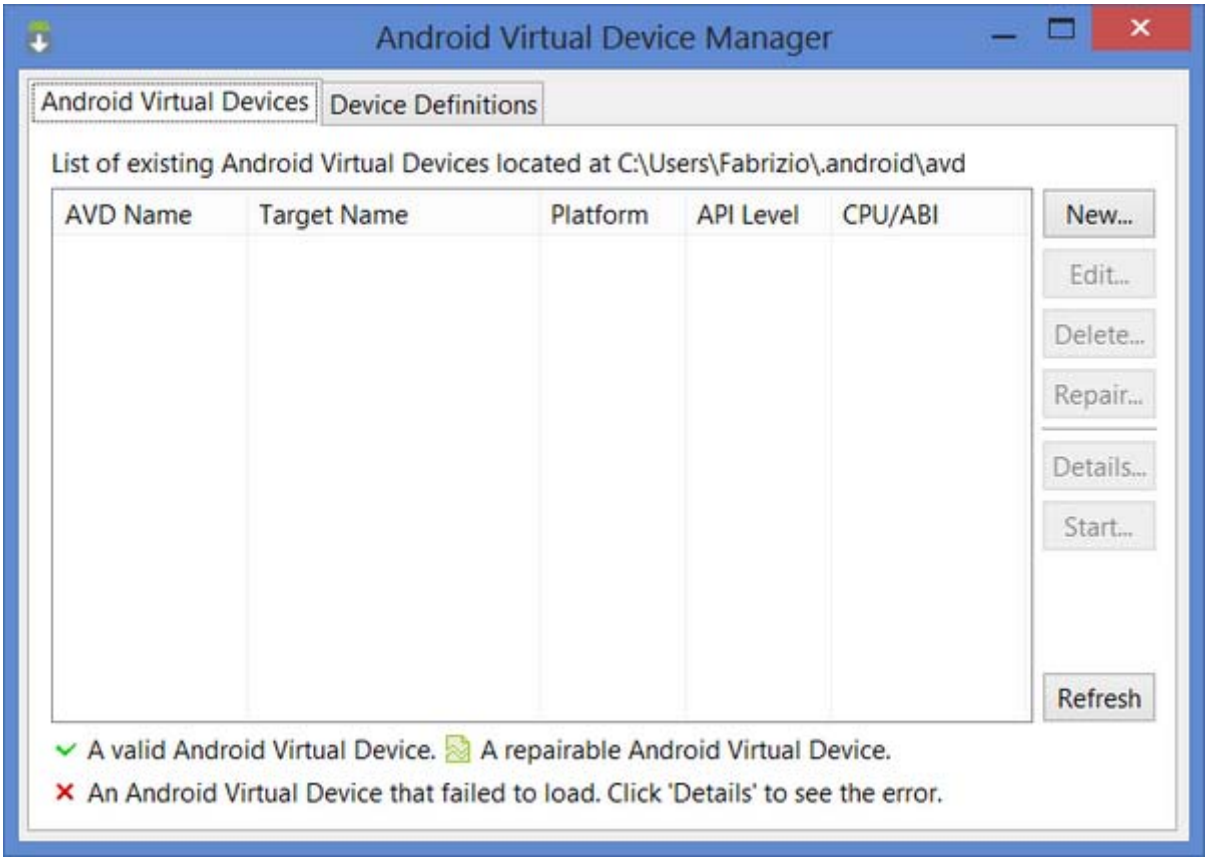
Whether you're using isCOBOL IDE or the command line, to run your app on the emulator you need to first create an Android Virtual Device (AVD).

An AVD is a device configuration for the Android emulator that allows you to model different devices.

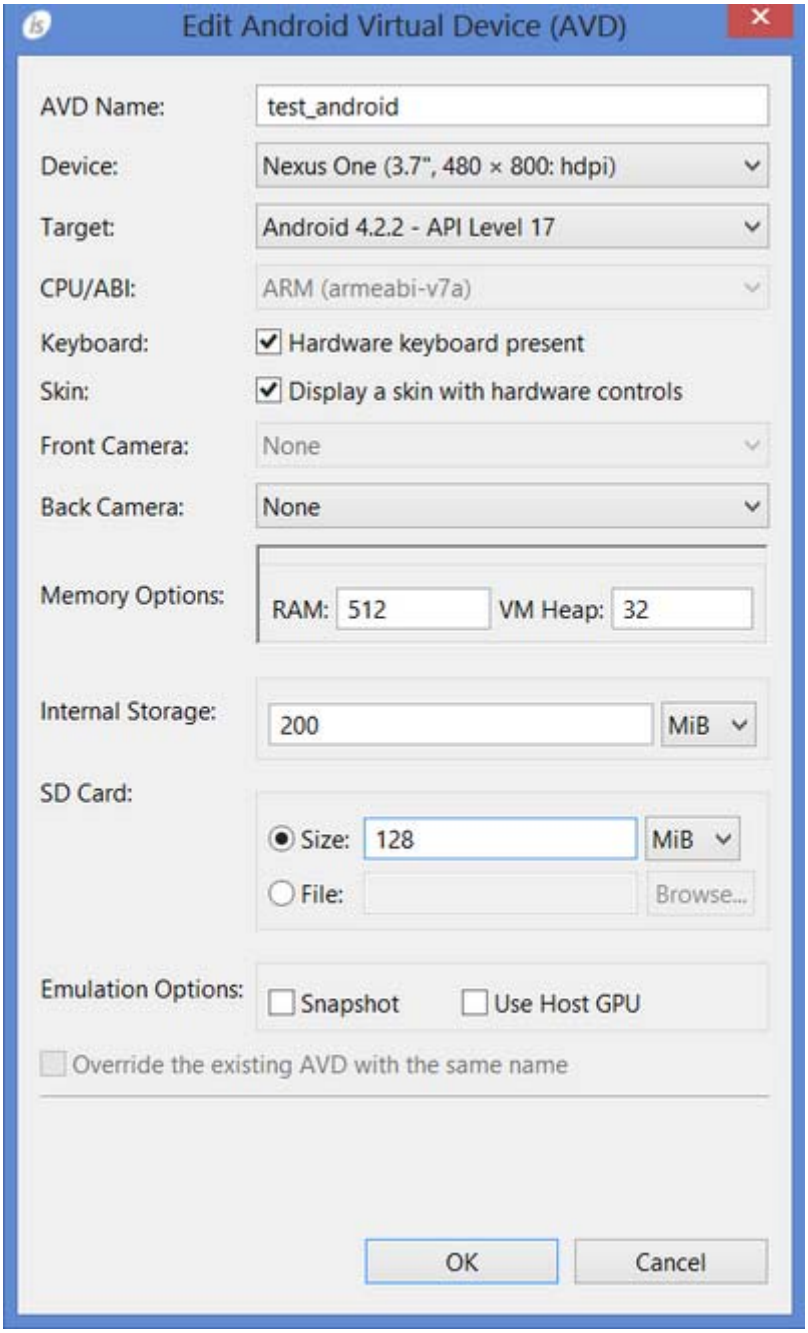
To create an AVD:

1. Launch the Android Virtual Device Manager:
   - You can find *AVD Manager.exe* in the SDK installation folder.
   - Alternatively, you can open a command prompt and execute:

```
android avd
```




2. In the Android Virtual Device Manager panel, click *New*.
3. Fill in the details for the AVD. Give it a name, a platform target, an SD card size, and a skin (HVGA is default). **Important -** Ensure to configure both internal memory and SD card as shown below:




From the AVD Manager you can start your virtual device as follows:

1. Select the new AVD from the Android Virtual Device Manager and click *Start*.
2. After the emulator boots up, unlock the emulator screen.

# Chapter 3

# Running the sample application

isCOBOL comes with a sample application named Password Reminder.

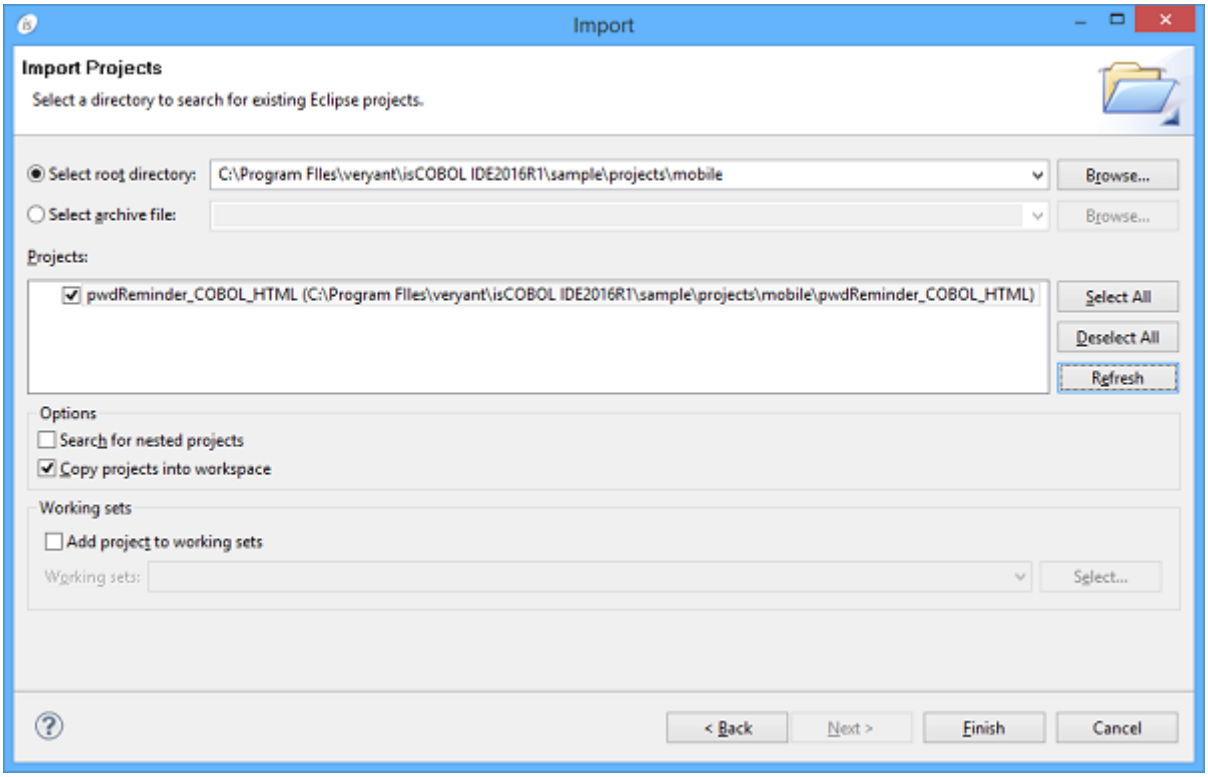
This chapter explains how to deploy and run the sample application.

## Running the sample application in isCOBOL IDE and exporting it to APK

isCOBOL IDE includes a sample project that shows how an isCOBOL Mobile application is done and how does it work.

Follow these steps to import the projects in the IDE and test them:

1. Create a new workspace or open an existing one
2. Click on the *File* menu
3. Choose *Import*
4. Choose *General* > *Existing Project into Workspace*
5. Having *Select root directory* checked, browse for the folder "C:\Program Files\Veryant\isCOBOL IDE2018R1\sample\projects\mobile"




6. Optionally check the option *Copy project into workspace* and click *Finish*

At this point you can run the sample application as EIS Servlet.

To run as EIS Servlet:

1. Right click on *pwdReminder_COBOL_HTML* in the Explorer tree
2. Choose *Run As* > *isCOBOL EIS Servlet*

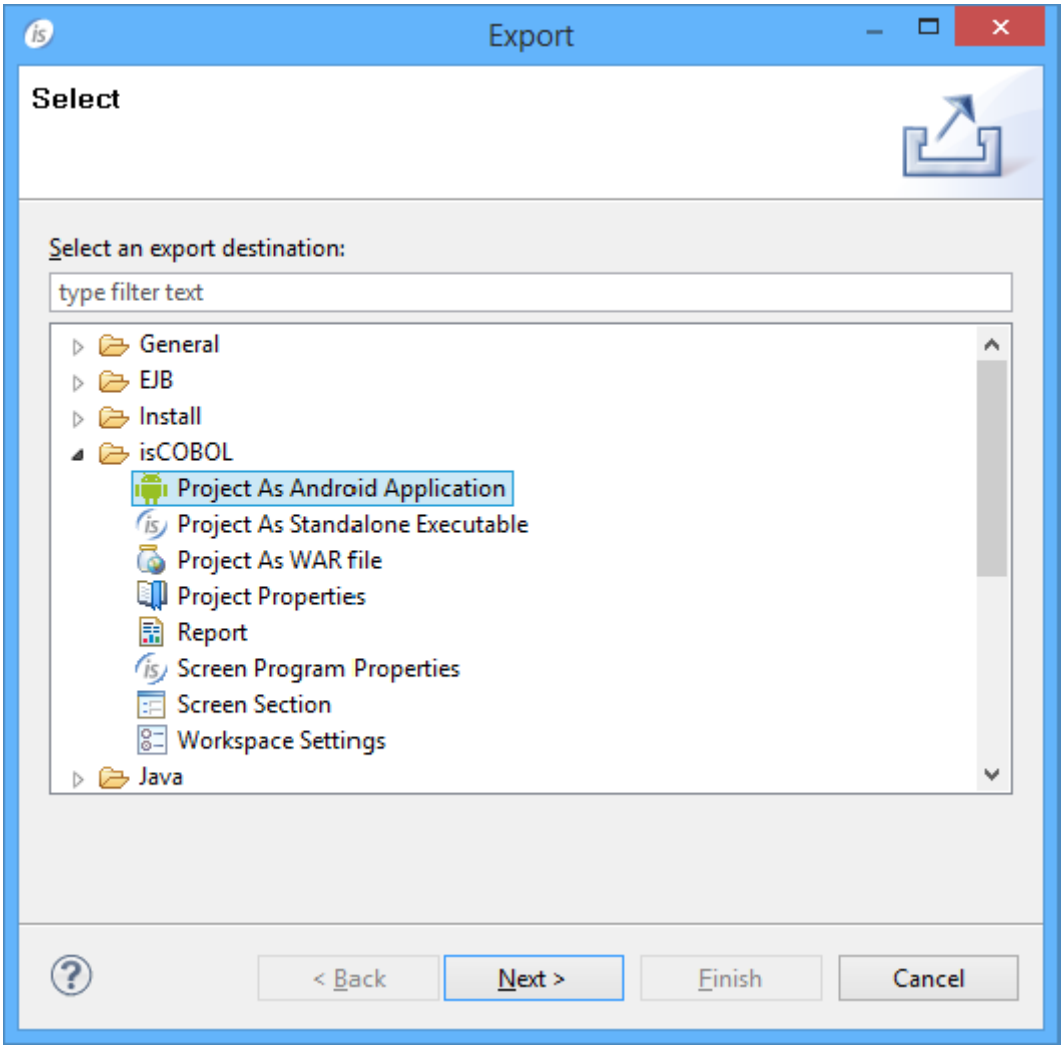
**Note -** for best rendering of the CSS styles it's suggested to use an external browser rather than the internal Eclipse browser. To use and external browser, before performing the above steps:

1. Click on *Window* in the menu bar
2. Choose *Preferences*
3. Choose *General* > *Web Browser* in the tree
4. Switch from "Use internal web browser" to "Use external web browser"
5. Click on the *Apply* button

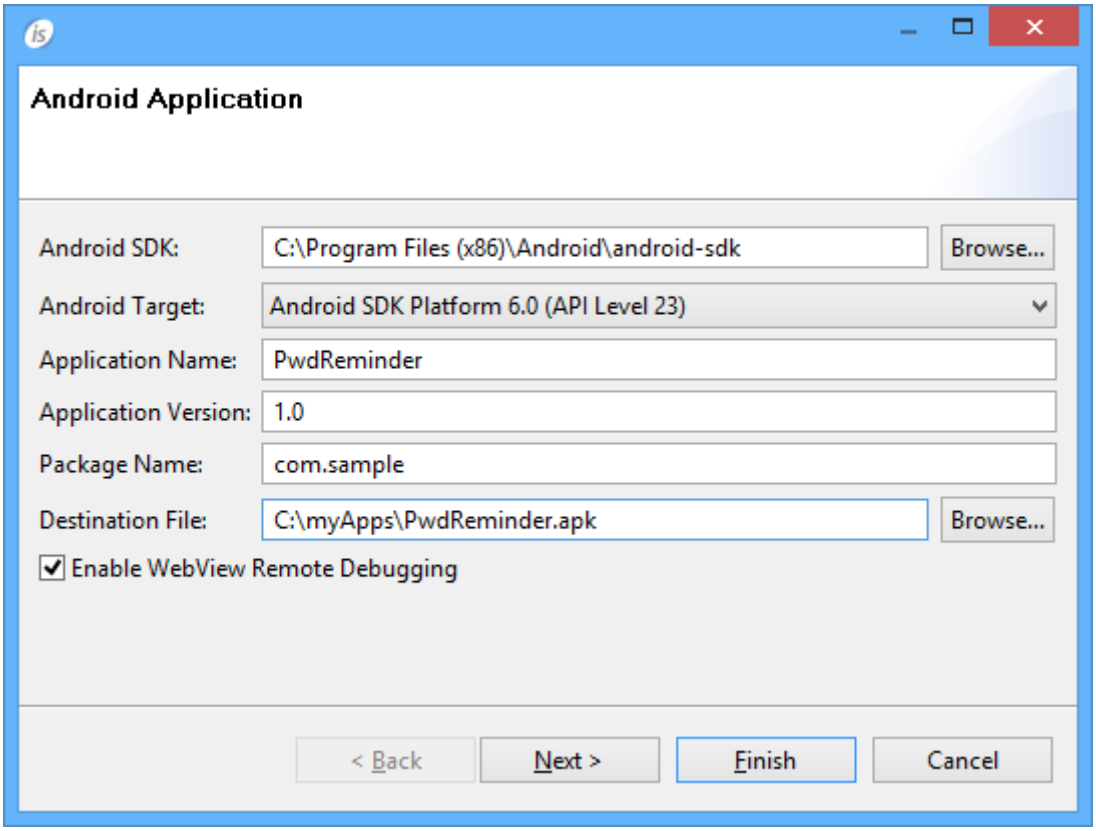
Once you've tested the application through the web-browser, it's time to make it and Android app.

1. Right click on *pwdReminder_COBOL_HTML* in the Explorer tree
2. Choose *Export*

3. Choose *Project as Android Application* from the *isCOBOL* tree




4. Fill the fields by providing the Android SDK location, th desired target version and the folder where the apk should be generated. Example:




5. click on the *Finish* button

See How to install and use the sample application for further information.

## Creating the APK of the sample application outside of isCOBOL IDE

The isCOBOL *samples* directory includes a folder named *mobile* that contains a folder named *pwd_Reminder* with all the necessary files to build and test the sample from command line..

1. Change to the directory *C:\Program Files\Veryant\isCOBOL2018R1\sample\mobile\pwd_Reminder*
2. Copy *ismobile.jar* from *C:\Program Files\Veryant\isCOBOL2018R1\mobile\lib* to *libs*
3. Compile the PASSWD program and put it in a jar named cobol.jar under the "libs" folder:

```
cd src
iscc PASSWD.cbl
jar -cf ..\libs\cobol.jar PASSWD.class
cd ..
```

4. Run the command

```
ant debug
```

See How to install and use the sample application for further information

# How to install and use the sample application

## Installation

The apk that you obtained can be tested in the AVD (Android Virtual Device, the emulator installed along with the SDK) as well as on a real physical Android device. For better performance and a more accurate outcome, the physical device is suggested.

In order to install the apk on a physical device, copy the apk to the device through USB or network file transfer features, then install it.

In order to install the apk on a AVD, proceed as follows:

1. start the AVD Manager with the command

```
android avd
```

   See The AVD Manager for details on how to configure and start an AVD

2. Run the command

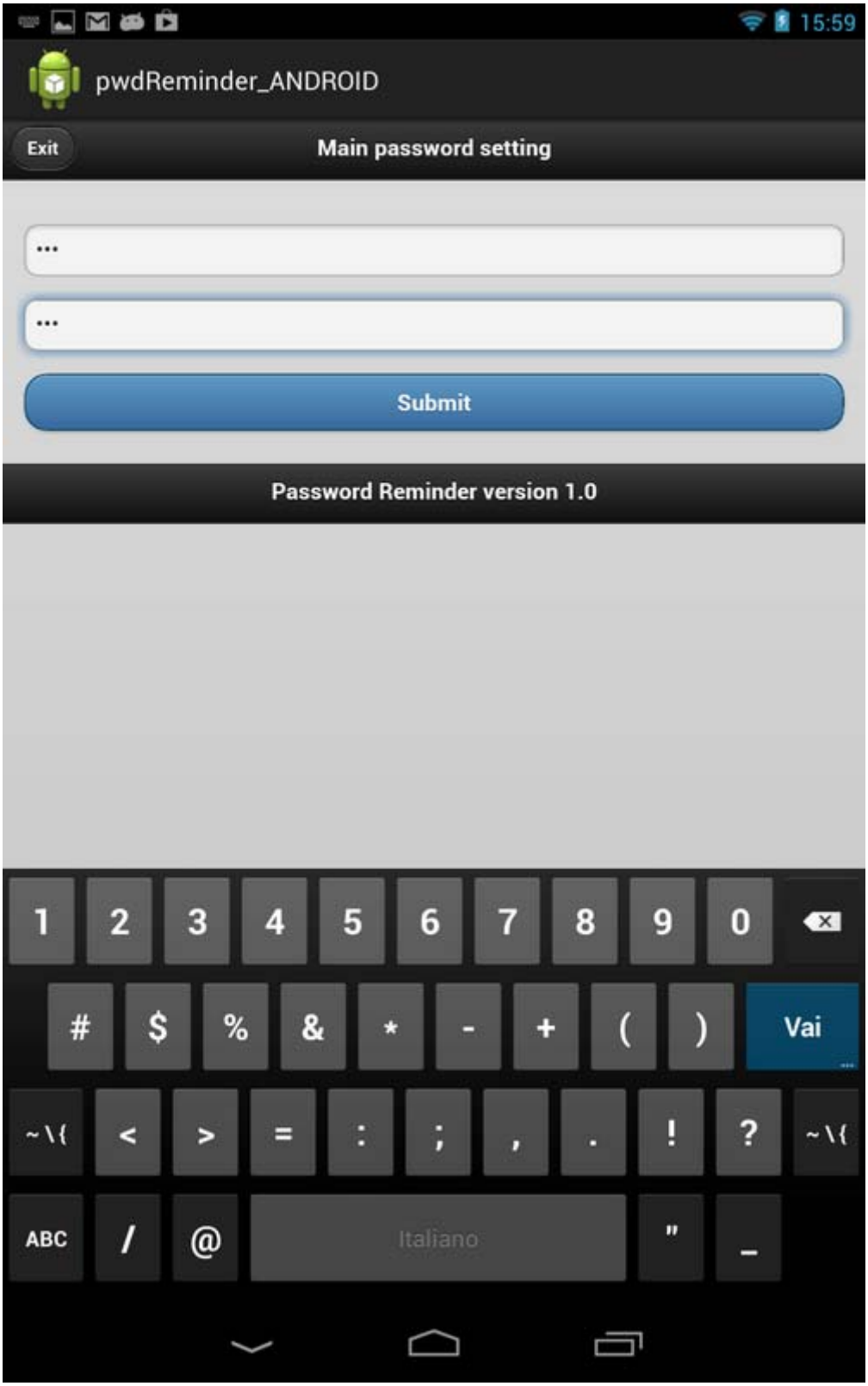
```
adb install \path\to\PwdReminder.apk
```

**Note -** the above command must be adjusted according to the location and name of your apk.

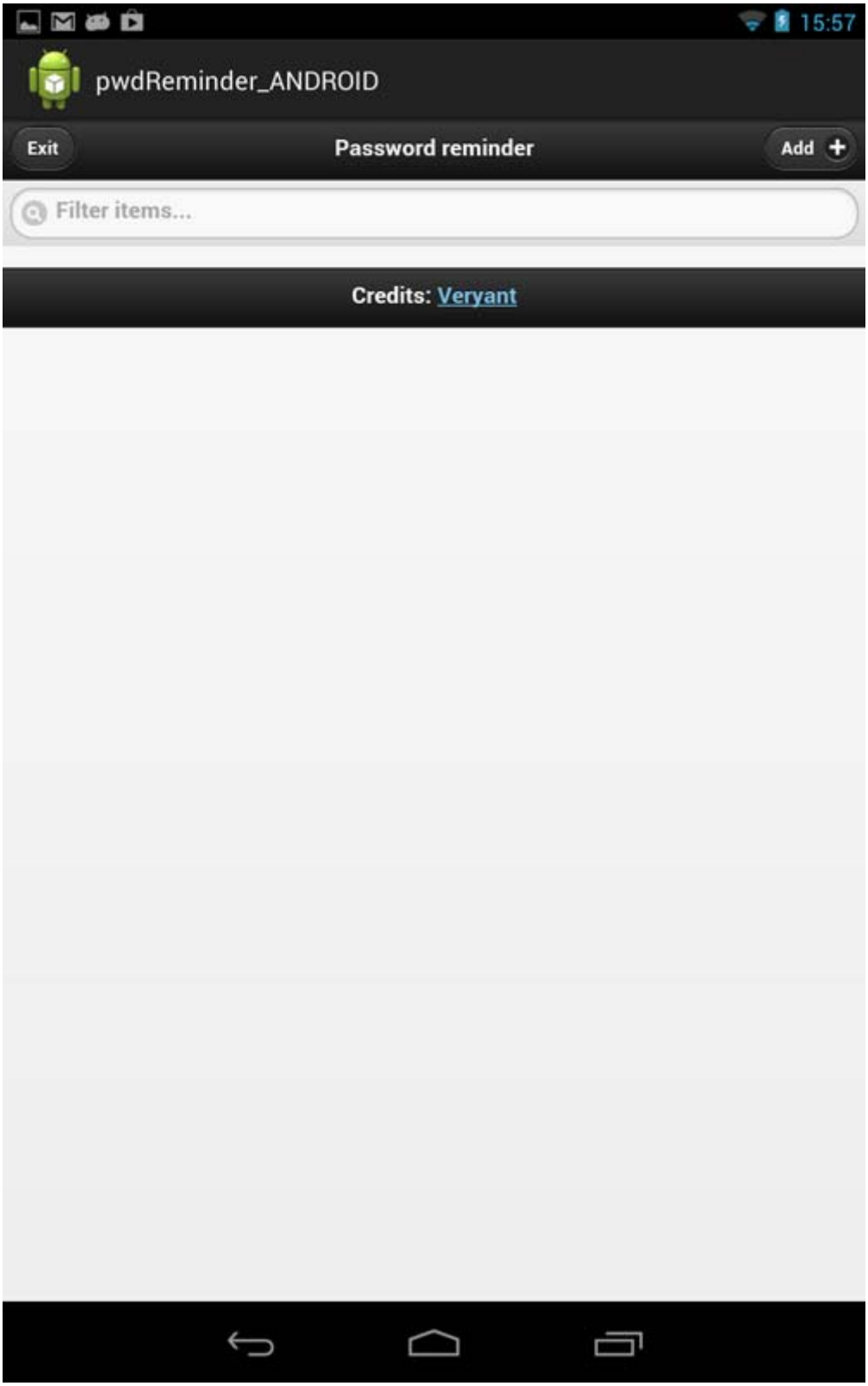
If the command is succesful, you will find a new app named *PwdReminder* in your AVD. Touch it to start.

## Usage

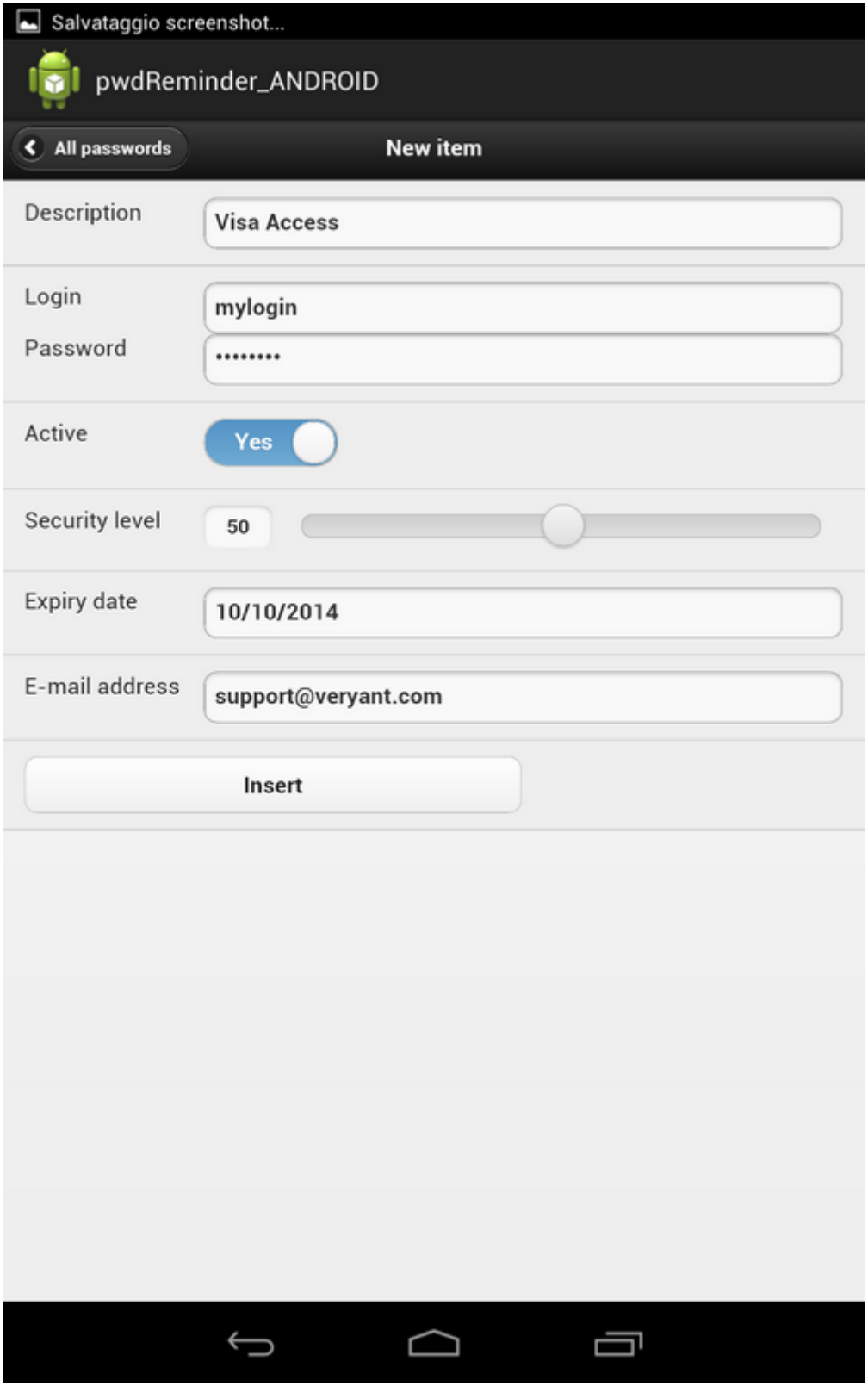
The following screen will appear in your emulator and you have to provide the main password to proceed.




1. Type the same text in both fields and touch *Submit* to reach the following screen:

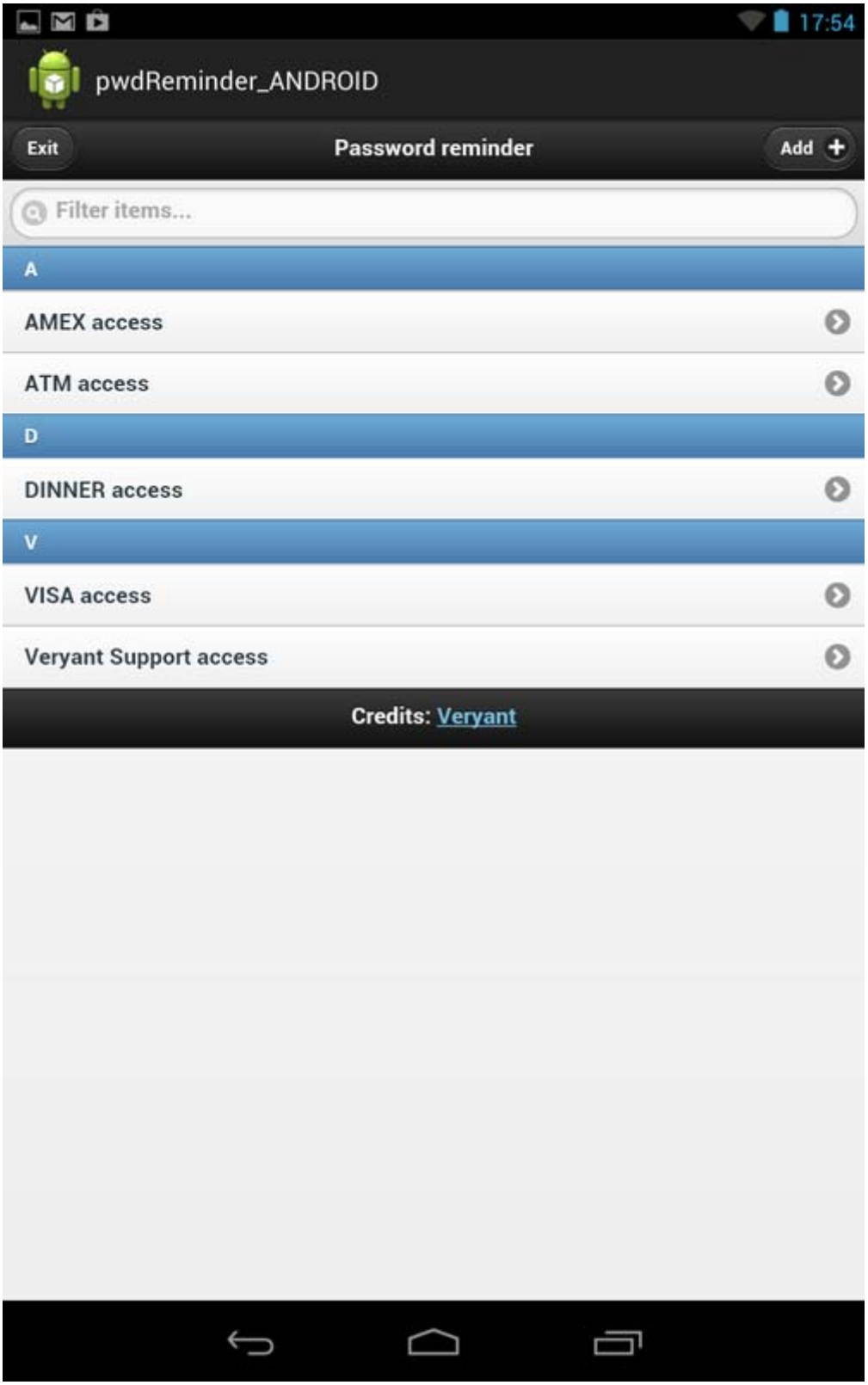



2. Touch the *Add* button to reach the "New item" screen where you can put free data, for example:




3. Touch the *All passwords* button to return to the list of passwords shown at step 1.

4. Repeat steps 2 and 3 to insert more items. At the end, the list of passwords should look like this:



# Chapter 4

# Development

## Developing an hello world application from scratch

The next chapter illustrates the steps to create an hello world application from scratch.

You have to produce and maintain two kind of source codes:

1. an HTML/JavaScript user interface
2. a COBOL program that receives requests from the HTML interface and returns a result to such interface

And you will be able to provide:

- a web application that can be used by any web-browser enabled device including Apple iOS devices
- an Android App that can run on Android devices natively

### Building the COBOL program

#### Using isCOBOL IDE

1. Click on the *File* menu
2. Choose *New*
3. Choose *isCOBOL Project*
4. Give it the name "simple_COBOL_HTML"
5. Click on the *Next* button until you reach the page where you can set *Compiler/Runtime options*
6. The class you're going to obtain may not be compatible with the destination Android device. In order to be compatible with more Android devices, including the oldest ones, it's good practice to generate classes that are backward compatible with older Java versions. For example, if you're using Java 8 or greater but you wish to obtain a class that is compatible also with Java 7 and Java 6, switch to the "J" page and fill the "-jo=" field as follows

```
"-source 1.6 -target 1.6"
```

7. It's also good practice to compile with the -whttp option in order to be advised if our program contains statements that are not supported in the Mobile environment. Switch to the "W" page and check the option "-whttp".
8. Click on the *Finish* button to confirm settings and complete the project creation.

**Note -** you will be prompted to switch to the isCOBOL HTML perspective. This switch is suggested in order to have a better view of this kind of project.

9. Right click on the *source* folder and choose *New*
10. Choose *Source File*
11. Give it the name "hello.cbl" and click on the *Finish* button.
12. Put the COBOL code into the program and compile it

### Without isCOBOL IDE

1. Create an empty text file and give it the name “hello.cbl”
2. Put the COBOL code into the file
3. Open a command prompt and change to the directory where the file is
4. It’s good practice to compile with the -whttp option in order to be advised if our program contains statements that are not supported in the Mobile environment.
   Compile the program with the command:

```
iscc -whttp hello.cbl
```

**Note -** The class you obtain may not be compatible with the destination Android device. In order to be compatible with more Android devices, including the oldest ones, it’s good practice to generate classes that are backward compatible with older Java versions. For example, if you’re using Java 8 or greater but you wish to obtain a class that is compatible also with Java 7 and Java 6, use this command:

```
iscc -whttp -jo="-source 1.6 -target 1.6" hello.cbl
```

5. HELLO.class will be created created; you need to include it in a jar to use it in the future steps. Use the following command:

```
jar -cvf cobol.jar HELLO.class
```

COBOL code

```
       PROGRAM-ID. hello.
       CONFIGURATION SECTION.
       REPOSITORY.
           class web-area as "com.iscobol.rts.HTTPHandler"
           .

       WORKING-STORAGE SECTION.
       01 hello-buffer identified by "_comm_buffer".
          03  filler identified by "_status".
              05 response-status pic x(2).
          03  filler identified by "_message".
              05 response-message pic x any length.
          03  filler identified by "hellotext".
              05  xml-hellotext pic x any length.

       LINKAGE SECTION.
       01  lnk-area object reference web-area.

       PROCEDURE DIVISION using lnk-area.
       main-logic.
           move "Operation successful" to response-message.
           move "OK" to response-status.
           move "Hello World from isCOBOL!" to xml-hellotext.
           lnk-area:>displayXML (hello-buffer).
           goback.
```

**Note -** The above code takes advantage of the HTTPHandler class (com.iscobol.rts.HTTPHandler) for communicating with the HTML user interface (explained later).

## Building the HTML interface and the web application

### Using isCOBOL IDE

1. Right click on the *html* folder
2. Choose *New* > *Other...*
3. Choose *Web* > *HTML File...*
4. Click on the *Next* button
5. Give it the name "index.html" and click *Finish*
6. Put the Content of Index.html in the file

At this point you can test the HTML application in the Eclipse to have a clue on how it will look on Android. So far you've produced a web application that can run on a web server and be used by any browser enabled device.

**Note -** Veryant suggests to run an external browser for the test. You can instruct the IDE to launch an external browser by following these simple steps:

1. Click on the *Window* menu
2. Choose *Preferences*
3. Expand *General* and select *Web Browser* in the tree
4. Check the "Use external web browser" option

5. Click *Ok*

To test the application, proceed as follow:

1. Right click on the project name in the tree
2. Choose *Run As* > *isCOBOL EIS Servlet*


If you wish to debug the COBOL program, choose *Debug As* > *isCOBOL EIS Servlet*, instead

**Note -** The debug of an 'EIS Servlet' consists in a remote debug session of the 'Jetty' application server included in the IDE. The first time the Debugger server suspends itself because it is waiting for a connection from the client; when the debugged program ends, the Debugger session doesn't terminate as it would happen in a stand alone Debugger session. The connection with the client part is still active. So the Debugger server goes in a 'continue' state, it means that it will suspend only when a breakpoint is reached. Hence, in order to debug the program another time, it is necessary to set some breakpoints.

Before going to the next step, that will make you run the simple hello world application on Android, it's necessary to provide a valid isCOBOL Mobile license key in the file *iscobol.properties* under the *resources* folder.

Edit the file *iscobol.properties* under the *html* folder and insert your license key. At the end, the file should look like this:

```
iscobol.mobile.license.2018=<your license code>
iscobol.exception.message=2
```

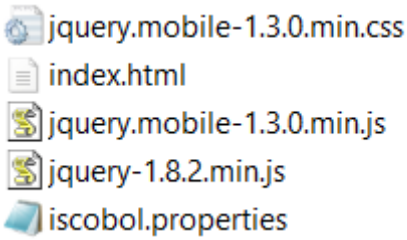
### Without isCOBOL IDE

1. Create a new file named "Index.html" and put the Content of Index.html in it.

2. Download JQuery scripts and css files (see Where to find JQuery files) .
3. Create a file named “iscobol.properties” and add the following entries to it:

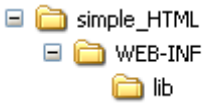
```
iscobol.mobile.license.2018=<your license code>
iscobol.exception.message=2
```

4. Using external software create a zip archive named "html.zip" and include the above items into it. The zip should contain:



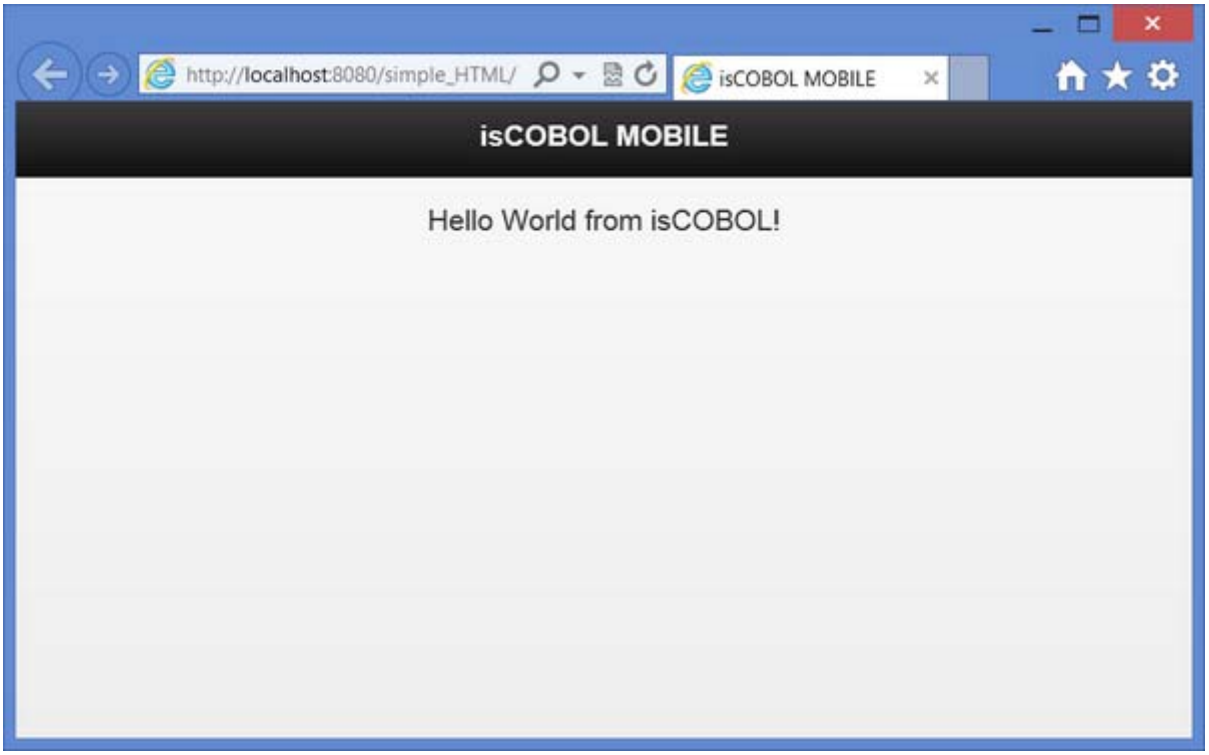

At this point you can test the application in a servelet container. The following steps show how to test in Tomcat.

1. Create the following folder structure under Tomcat’s *webapps* directory:




2. Put the following files under *simple_HTML*:
   - index.html
   - jquery-1.8.2.min.js
   - jquery.mobile-1.3.0.min.css
   - jquery.mobile-1.3.0.min.js
3. Create a file named “web.xml” under *WEB-INF* and put the Content of Web.xml into it.
4. Put the following files under *lib*:
   - cobol.jar (previously created)
   - iscobol.jar (found in C:\Program Files\Veryant\isCOBOL2018R1\lib)
5. Start the Tomcat service
6. Assuming that you’re testing on localhost and Tomcat is started on the default port, navigate to “http://

127.0.0.1:8080/simple_HTML”



- Content of Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>testHTML</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <filter>
        <filter-name>isCOBOL filter</filter-name>
        <filter-class>com.iscobol.web.IscobolFilter</filter-class>
  </filter>
  <filter-mapping>
        <filter-name>isCOBOL filter</filter-name>
        <url-pattern>/servlet/*</url-pattern>
  </filter-mapping>
  <servlet>
        <servlet-name>isCobol</servlet-name>
        <servlet-class>com.iscobol.web.IscobolServletCall</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>isCobol</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
  <listener>
   <listener-class>com.iscobol.web.IscobolSessionListener</listener-class>
  </listener>
</web-app>
```

Content of Index.html

```
<html>
    <head>
    <title>Test Mobile </title>
   <link href="jquery.mobile-1.3.0.min.css" rel="stylesheet" type="text/css" />
   <script src="jquery-1.8.2.min.js"></script>
   <script src="jquery.mobile-1.3.0.min.js"></script>


   <script>
   function handleError (jqXHR, textStatus, errorThrown) {
      alert (textStatus +" "+ jqXHR.status + " "+jqXHR.statusText +
                  "\n" + jqXHR.responseText);
   }
   function handleSuccess (data, textStatus, jqXHR) {
      response = jqXHR.responseText;
      try {
         xmlDoc = jQuery.parseXML (response);
      } catch (err) {
         alert (response);
         return false;
      }
      xml = jQuery (xmlDoc);
      _status = xml.find ("_status");
      _message = xml.find( "_message" );
      _hello = xml.find( "hellotext" );
      jQuery("#hello_div").html(_hello.text());
      return true;
   }

   function callServer (cobolProg) {
      var url = "servlet/isCobol(" + cobolProg + ")";
      jQuery.ajax(url, {
                    success: handleSuccess,
                    error: handleError
                });
      return false;
   }

   window.onload = callServer("HELLO");
   </script>
```

```
</head>
<body>

<div data-role="page">

    <div data-role="header" data-theme="a">
        <h1>isCOBOL MOBILE</h1>
    </div><!-- /header -->

    <div data-role="content" data-fullscreen="true">
        <div id="hello_div" align="center"></div>
    </div><!-- /content -->

</div><!-- /page -->

</body>
</html>
```

### Where to find JQuery files

JQuery script and css files used by Index.html can be downloaded from the following sites:

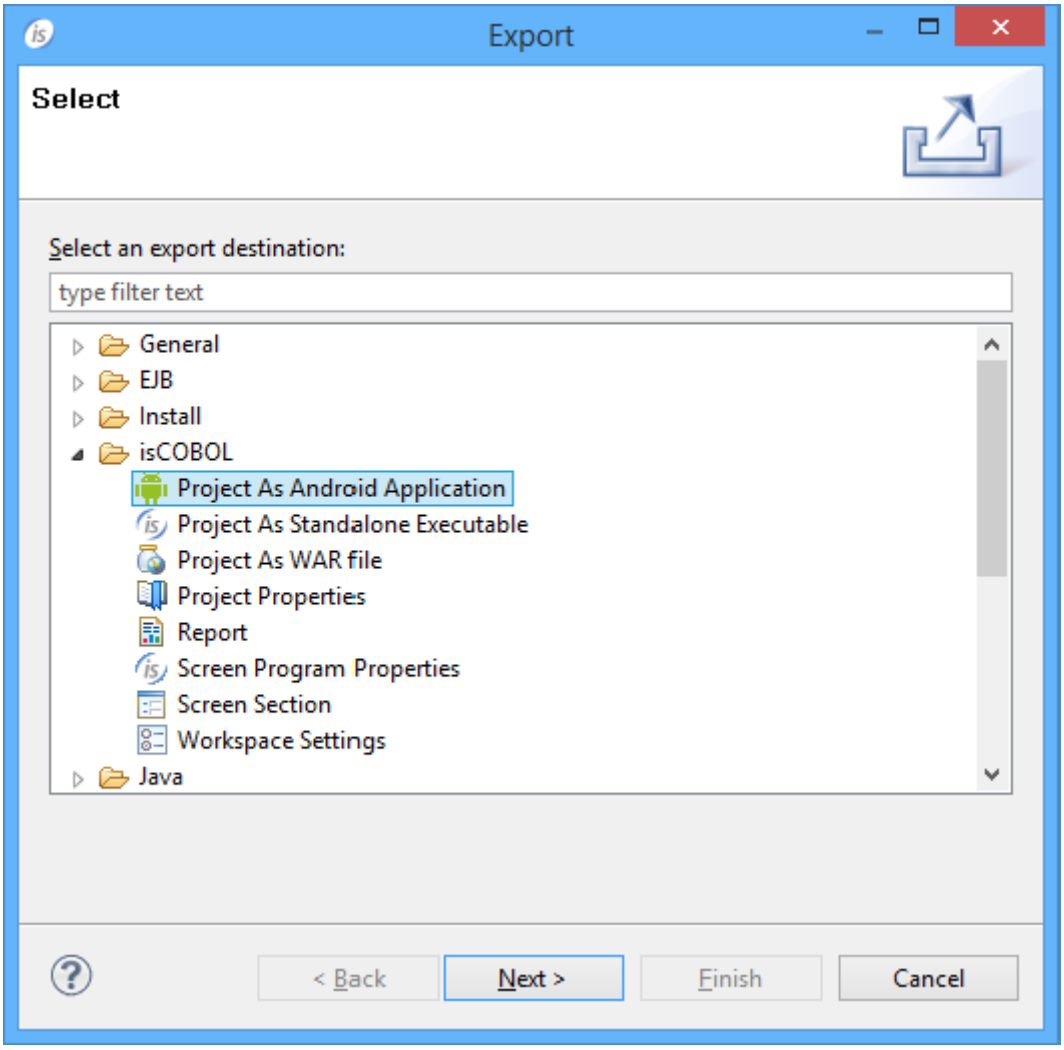
| File | Web Site | Direct URL |
|---|---|---|
| jquery-1.8.2.min.js | http://jquery.com | http://code.jquery.com/jquery-1.8.2.min.js |
| jquery.mobile-1.3.0.min.css | http://jquerymobile.com/ | http://code.jquery.com/mobile/1.3.0/jquery.mobile-1.3.0.min.css |
| jquery.mobile-1.3.0.min.js | http://jquerymobile.com/ | http://code.jquery.com/mobile/1.3.0/jquery.mobile-1.3.0.min.js |

## Creating and testing the Android App

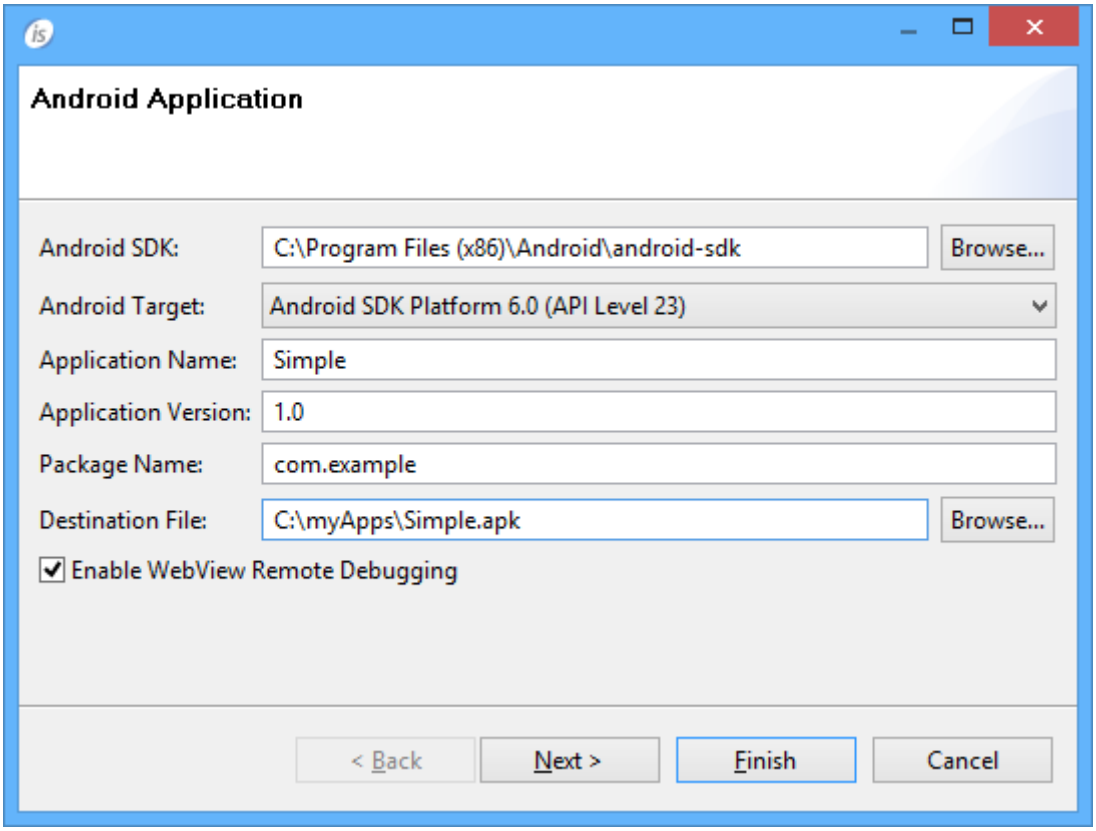
### Using isCOBOL IDE

1. Right click on *simple_COBOL_HTML* in the Explorer tree
2. Choose *Export*

3. Choose *Project as Android Application* from the *isCOBOL* tree




4. Select the desired project from the list, then click *Next*
5. Provide the required information: Android SDK location, target version, package and the folder where the generated apk should be saved. Example:

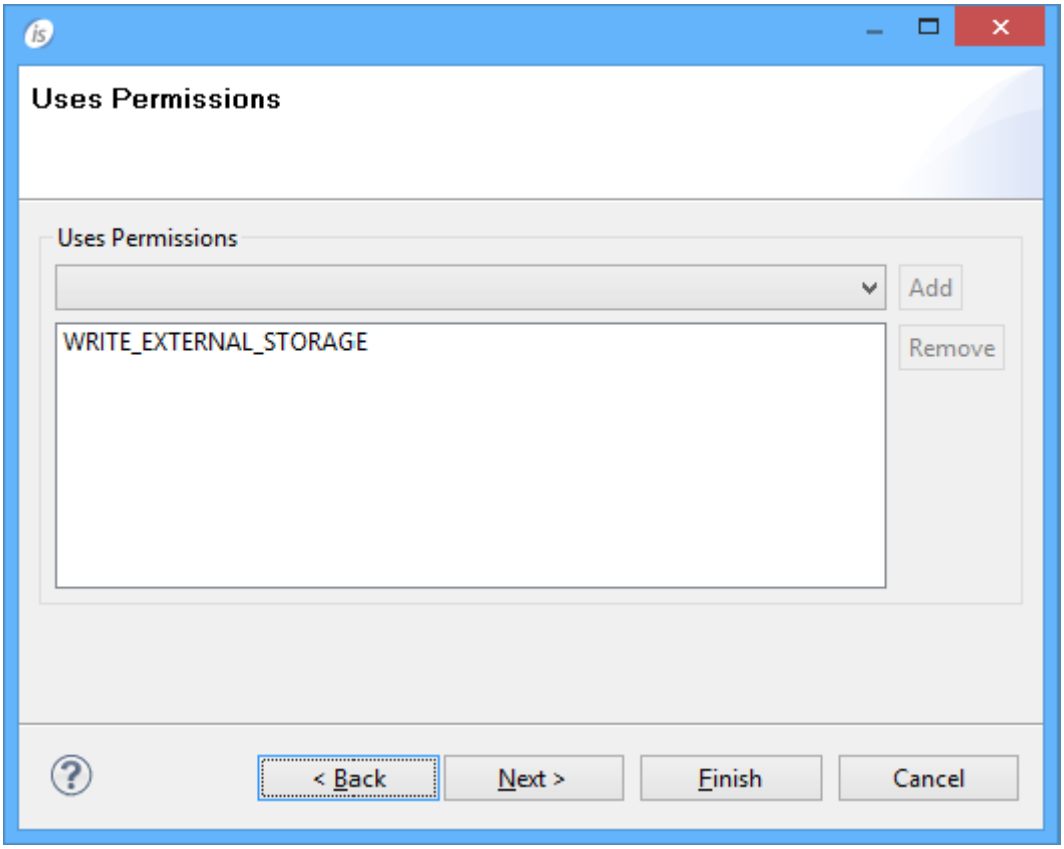



**Note** - Do not use the same package for more applications. The package is a sort of unique ID in the Android system.

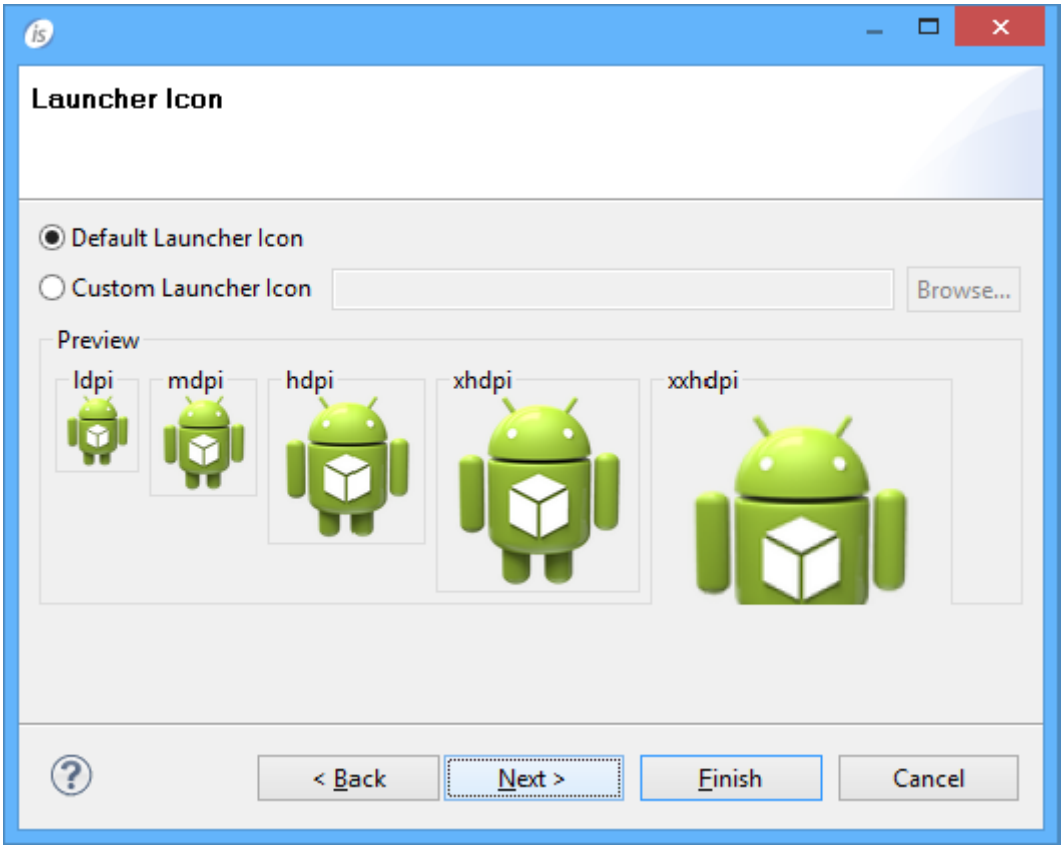
The "Enable WebView Remote Debugging" option enables the ability to attach the app with a Chrome browser in order to debug the HTML/JS frontend as described here.

6. You can click finish or go ahead with the following optional steps.

7. The next step allows to customize permissions. By default, only WRITE_EXTERNAL_STORAGE is active.

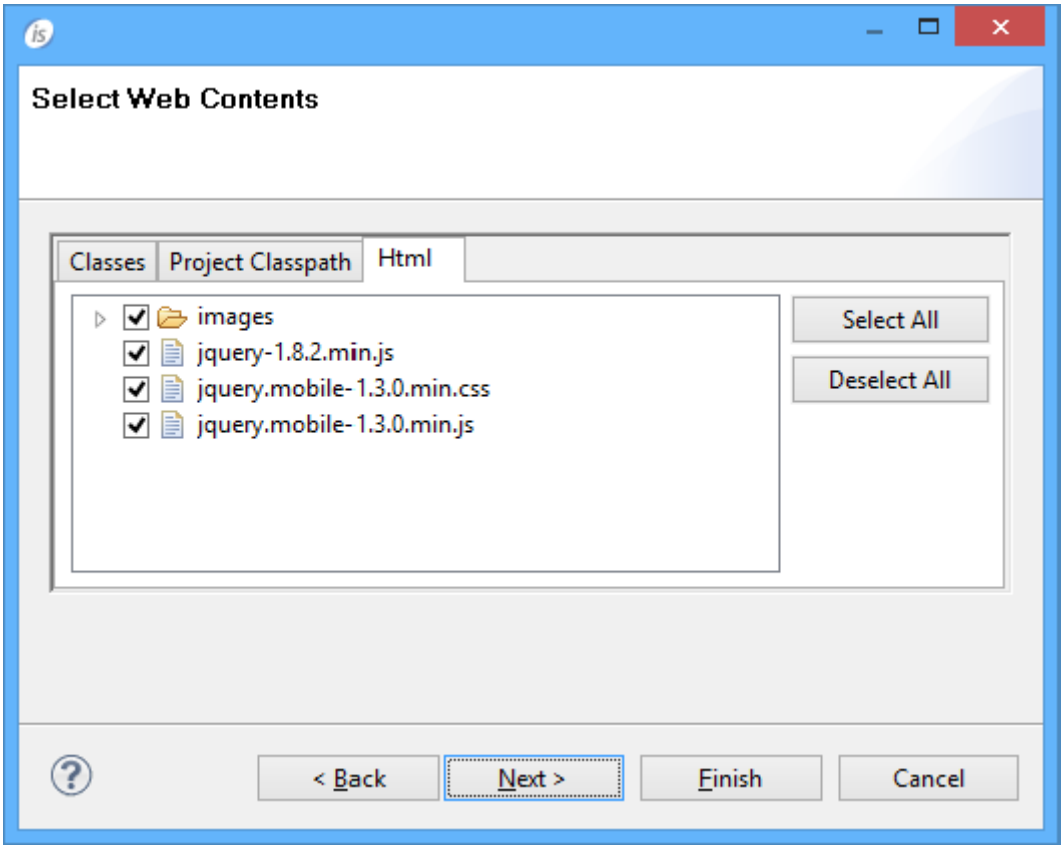


8. The next step allows you to configure the application icon



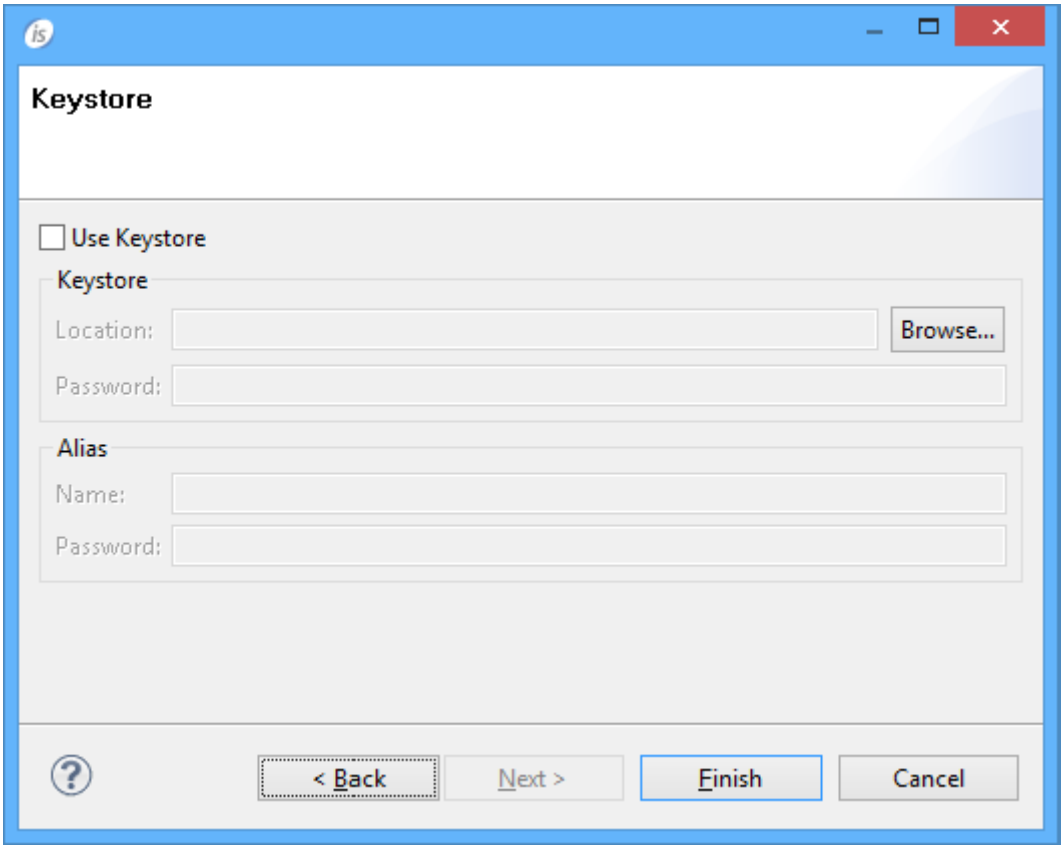

9. The next step allows you to exclude some of the project items (e.g. a program or a css file). Ensure that these

items are really useless before excluding them, otherwise your app may not work correctly.




10. If you have a valid keystore to provide or you wish to exclude some project files from the app, click *Next*,

otherwise click *Finish*.




**Note -** If no keystore is provided, the IDE will include a default one. The default keystore is suitable for testing purposes. Before publishing your app in the Marketplace, you should apply a valid keystore to it.

Onc e the apk has been generated, you can test it either in the AVD (Android Virtual Device, the emulator installed along with the SDK) or in a real physical Android device.

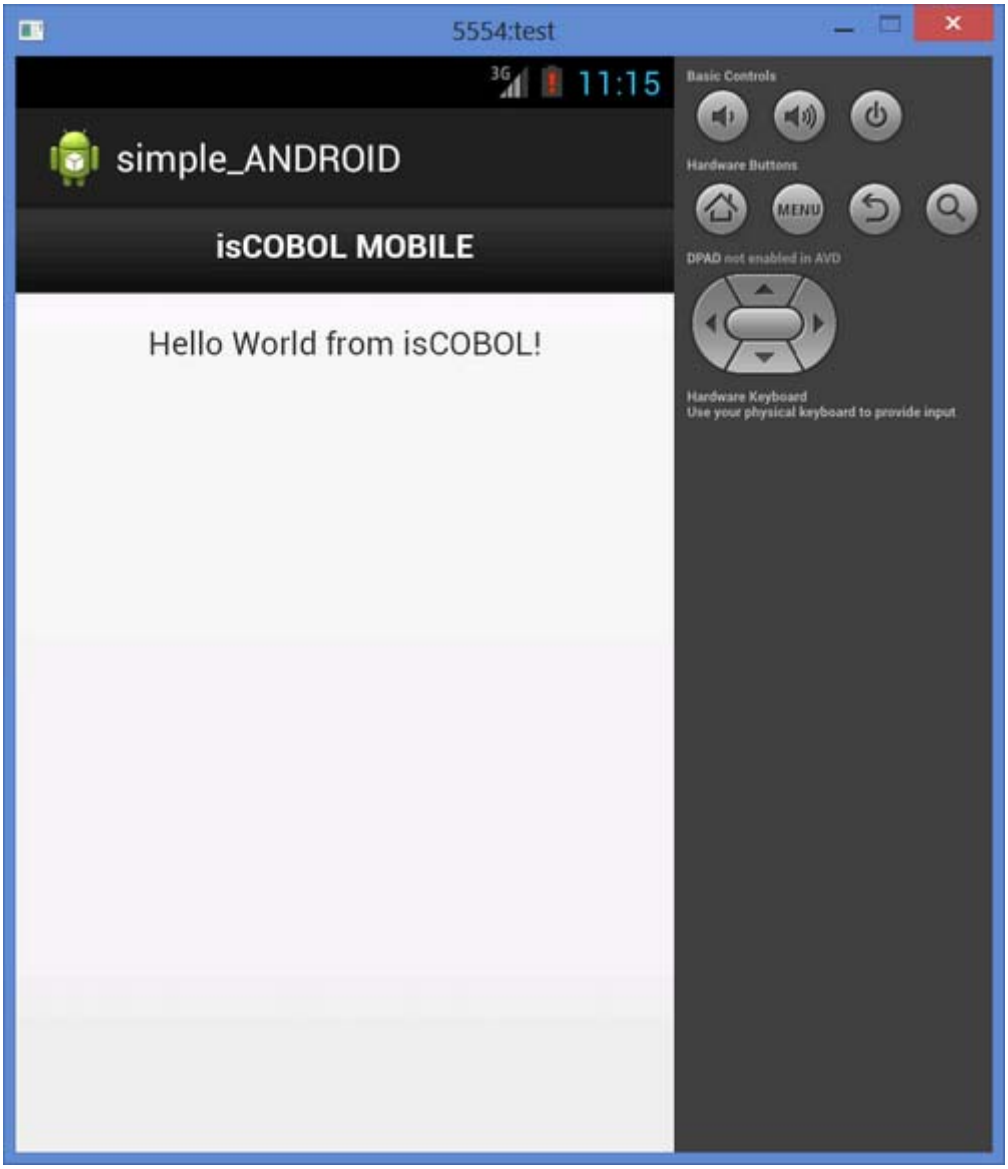
In order to test it in a real device, copy the apk to the device through USB or network file transfer features and install it.

In order to test it in the AVD, run the following command:

```
adb install path/to/Simple.apk
```

**Note -** the above command must be adjusted according to the location and name of your apk.

The above command installs the app in the emulator and starts it. Find your app in the emulator screen and touch it to start it:




### Without isCOBOL IDE

1. Create an empty folder to host project files (e.g. "C:\android_test")
2. Run the command:

```
android create project --target 1 --name simple_ANDROID --path c:\android_test --
   activity MainActivity --package  com.example.simple_android
```

You should obtain the following output

```
Created directory C:\android_test\src\com\example\simple_android
Added file c:\android_test\src\com\example\simple_android\MainActivity.java
Created directory C:\android_test\res
Created directory C:\android_test\bin
Created directory C:\android_test\libs
Created directory C:\android_test\res\values
Added file c:\android_test\res\values\strings.xml
Created directory C:\android_test\res\layout
Added file c:\android_test\res\layout\main.xml
Created directory C:\android_test\res\drawable-xhdpi
Created directory C:\android_test\res\drawable-hdpi
Created directory C:\android_test\res\drawable-mdpi
Created directory C:\android_test\res\drawable-ldpi
Added file c:\android_test\AndroidManifest.xml
Added file c:\android_test\build.xml
Added file c:\android_test\proguard-project.txt
```

3. In the directory you created at step 1, edit the file *AndoirdManifest.xml* and replace the current content with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.simple_android"
    android:versionCode="1"
    android:versionName="1.0"
    android:installLocation="auto" >
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name="com.example.simple_android.MainActivity"
            android:label="@string/app_name"
            android:configChanges="keyboardHidden|orientation|screenSize" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

The manifest file describes the fundamental characteristics of the app and defines each of its components. Our main goal is to obtain write permissions on the device in order to store the html files for the UI and let the COBOL program create files if necessary.

4. Switch to the *res\layout* sub folder, rename the file *main.xml* to *activity_main.xml* and replace the current content with the Content of main_activity.xml

5. Switch to the *src\com\example\simple_android* sub folder and edit the file *MainActivity.java* replacing the current content with the Content of MainActivity.java.
6. Add *ismobile.jar* (taken from C:\Program Files\Veryant\isCOBOL2018R1\mobile\lib) and *cobol.jar* (previously produced) to the *libs* sub folder.
7. Create a folder named "raw" under the *res\layout* sub folder and put *html.zip* (previously created) into that folder.
8. (optional) Edit the file *strings.xml* under the *res\values* subfolder and provide a custom name for your app. The current name is "MainActivity", you might call it "simple_ANDROID" to make it match with the project name or you can use whatever name you prefer.
9. From the root directory of your project (e.g. C:\android_test) run the command:

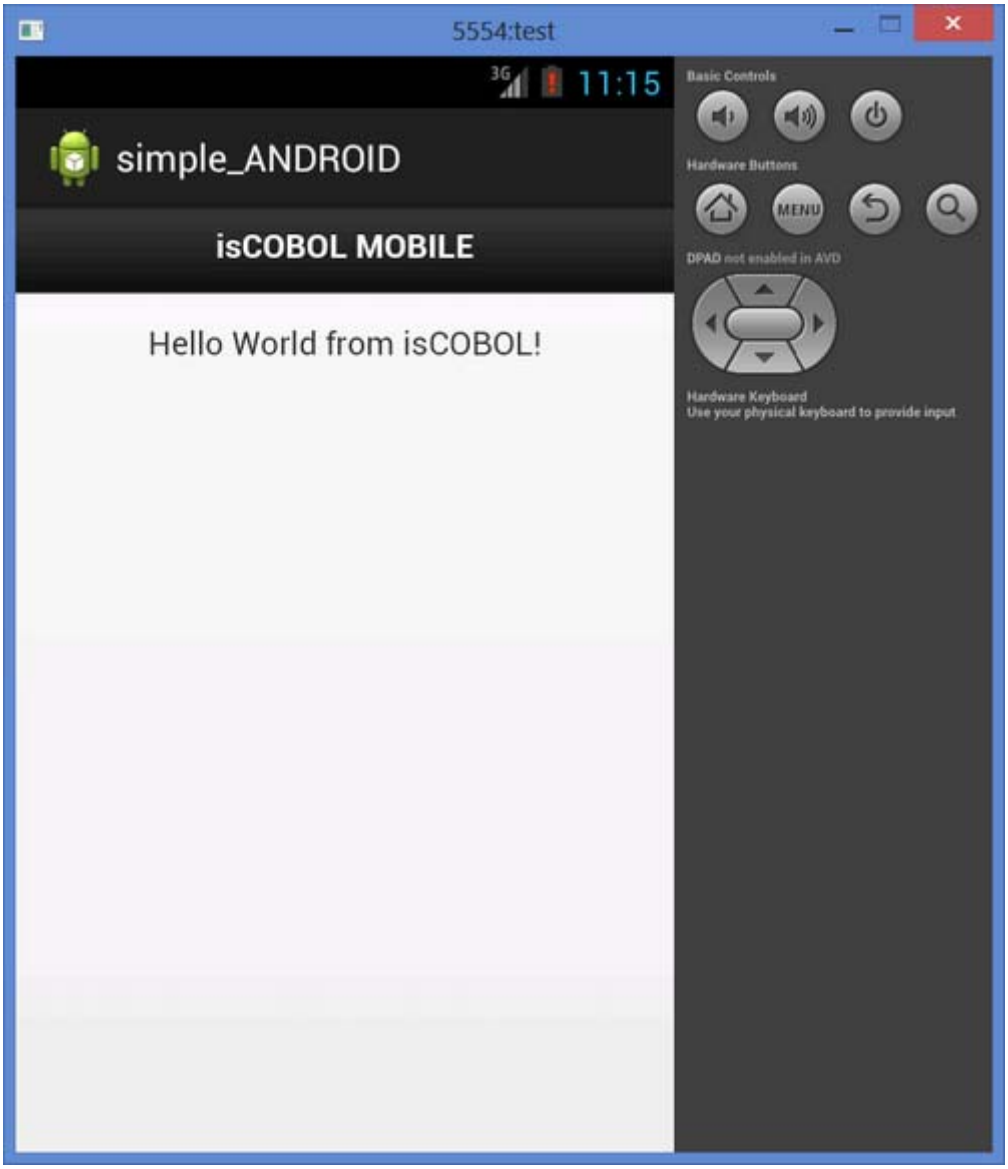
```
ant debug
```

   The above command builds the apk file of your app

10. If you haven't done yet, create and configure an emulator as epxlained at The AVD Manager .
11. Start the emulator.
12. From the root directory of your project (e.g. C:\android_test) run the command:

```
adb install bin/simple_ANDROID-debug.apk
```

   The above command installs the app in the emulator and starts it. Find your app in the emulator screen and touch it to start it:




**Note -** for better performance and a more accurate result, you can copy the apk on a real physical Android device and test it there.

Content of main_activity.xml

```
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

The WebView control allows to render the HTML user interface of our application. This is the only control we need.

Content of MainActivity.java

```
package com.example.simple_android;

import java.io.PrintWriter;
import java.io.StringWriter;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

import android.os.Bundle;
import android.webkit.WebView;

import com.iscobol.iscobol4android.IsCobolMainActivity;

public class MainActivity extends IsCobolMainActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      final WebView webView = (WebView) findViewById(R.id.webView1);
      init (webView, R.raw.html);
   }
}
```

# Chapter 5

# Troubleshooting

This chapter explain how to diagnose errors that may appear while working with isCOBOL Mobile.

## How to debug the HTML interface

Problems in the HTML and Javascript code can be monitored and debugged through web-browsers advanced developers features. Consult your web-browser documentation for details about the availability and the usage of such features.

## Issues during the Export to Android application

When the IDE exports a project to Android application it prints the outcome in the *Console* view. Some errors may be available in the *Error log* view instead.

## AVD and LogCat

The Android SDK provides a useful tool named LogCat that allows to collect log and debug information. Such information is filtered and shown as output. This tool is useful to diagnose startup and runtime problems of your app once it has been installed in the AVD.

You can start LogCat from the command-line using the command:

```
adb logcat
```