Voice Control SDK

Vuzix M100 Developer SDK

Ron DiNapoli
Vuzix Corporation
Created: July 22, 2015
Last Update: July 22, 2015

Introduction

The Vuzix Voice Control SDK allows you to add the ability to navigate your Smart Glasses application with voice commands. A good voice command system will give your users the freedom to make the most of your application while leaving their hands free to perform job related tasks unencumbered by a "hands on" interface.

When writing a voice controlled application, you have two choices. The first choice is to use the Google SpeechRecognizer API (android.speech.SpeechRecognizer). The Google SpeechRecognizer API acts as an abstraction layer to multiple different speech recognition engines. Most of these engines require an active Wifi connection as the actual speech recognition process is performed in the cloud. The native speech recognition engine provided by Vuzix on our Smart Glasses (which performs recognition on a finite set of commands locally and does not use the cloud) is also interfaced via the Google APIs enabling you to have the most flexibility possible. The second choice you have is to access the native speech recogniztion capabilities provided by Vuzix directly. This involves using the Vuzix VoiceControl class (com.vuzix.speech.VoiceControl) directly. The benefit here is that it cuts out the abstraction layer and simplifies your code. When implementing a simple set of voice commands to navigate an application, this is the preferred method.

This document focuses on using the native Vuzix VoiceControl class in your application. For information on using the Google APIs, please consult the Google Android SDK documentation for android.speech.SpeechRecognizer.

About Vuzix VoiceControl

Vuzix VoiceControl utilizes a local speech recognition engine—the speech is processed right on the M100 and is never shipped out to the cloud for processing! To minimize the resources required for this operation we provide a number of small "grammars" – each of which contain a number of words that can be recognized by your application. The default grammar contains basic commands that are used to control the Smart Glasses from the Home Screen. These commands can be useful in other applications as well. Additionally, we make grammars available containing words suitable for the following areas: medical, warehousing, media, camera and navigation.

When using Vuzix VoiceControl, the "base" or "default" grammar is loaded by default. This grammar contains the following words/phrases:

- move (left/right/up/down)

- go (back/home/left/right/up/down)
- scroll (left/right/up/down)
- call back
- set (clock/time)
- next
- previous
- forward
- halt
- select (#)
- complete
- launch (#)
- cancel
- stop
- exit
- menu
- volume (up/down)
- mute
- confirm
- call
- dial
- hang up
- answer
- ignore
- end
- redial
- contacts
- favorites
- pair
- unpair
- sleep
- shut down
- <#>
- cut
- copy
- paste
- delete
- voice (on/off)
- show help

When you start a voice recognition process in your application, these words will be identified and reported to you by default.

You can also load additional grammars and the Vuzix VoiceControl system will recognize the "union" of the word sets for each grammar.

Information on how to load the other provided grammars and how to work with us to create a custom grammar for your application is mentioned later in this document.

Using Vuzix VoiceControl

Implementing voice control in your M100 application is a relatively simple process. You will need to create a new class that extends the `com.vuzix.speech.VoiceControl` base class (available through the SDK), know when to turn speech recognition "on" and "off" (programmatically) and

override the `onRecognition()` method to receive data on commands that were recognized.

Defining your own voice control class can be as simple as this:

```
import com.vuxiz.speech.VoiceControl;
import android.widget.Toast;

public class MyVoiceControl extends VoiceControl {
     public MyVoiceControl(Context context) {
          super(context);
     }
     protected void onRecognition(String result) {
          Toast.makeText(this,"Received: "+result,
                    Toast.LENGTH_SHORT).show();
     }
}
```

This class can be in a separate .java file or can be nested inside another class.

When creating an Android Activity that is going to use voice recognition, it is customary to allocate an instance of your `VoiceControl` class in that Activity's `onCreate()` method. This would look something like this:

```
public class MainActivity extends Activity {
     private MyVoiceControl mVC;

     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);

          // Set up custom voice control class
          mVC = new MyVoiceControl(this);

          // Make sure we were able to create mVC
          if (mVC == null) {
               Toast.makeText(this,"Can't create mVC",
                    Toast.LENGTH_SHORT).show();
               return;
          }

          // Turn voice control on
```

```
            mVC.on();

            // Rest of onCreate() code below…
      }
}
```

The above example code would allocate an instance of your custom voice control class and turn on voice recognition whenever your activity is created.

An important aspect of the voice control mechanism to keep in mind is that it can only be used by one task at a given time. Since the M100's home screen/launcher application is also voice-enabled, it is important to turn off voice control when your application looses focus to avoid any issues/errors that may result from having two applications attempting to use the voice control mechanism at the same time. This is best handled by overriding the `onPause()` and `onResume()` methods for the main activity in your application. Also, to play it safe, you should turn off voice control in the activity's `onDestroy()` method in the event that the activity is destroyed without `onPause()` being called first.

```
public class MainActivity extends Activity {
      private MyVoiceControl mVC;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            // see implementation above
      }

      @Override
      protected void onPause() {
            if (mVC != null) {
                  mVC.off();
            }
            super.onPause();
      }

      @Override
      protected void onResume() {
            if (mVC != null) {
                  mVC.on();
            }
            super.onResume();
      }

      @Override
      protected void onDestroy() {
            if (mVC != null) {
```

```
            mVC.off();
        }
        super.onDestroy();
    }

}
```

One final configuration you must make in your code before being able to use voice control is to declare that your app requires the RECORD_AUDIO permission. This is because the M100 will need to record audio and then pass the recorded audio on to the speech recognition engine. To specify this permission, add the following line of text to your main `AndroidManifest.xml` file, just *before* the application tag:

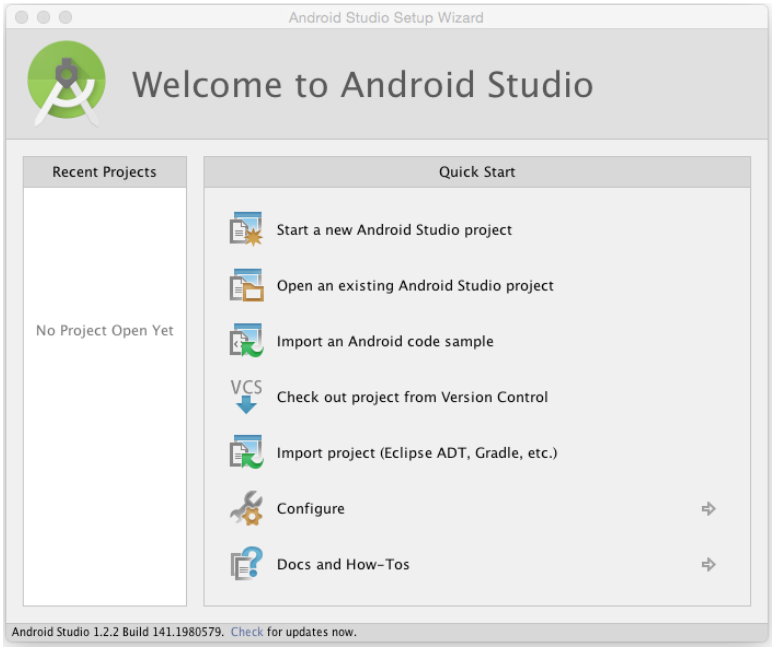
<uses-permission android:name="android.permission.RECORD_AUDIO" />

This covers the basic operation of the Vuzix voice control mechanism. Advanced features such as switching/adding provided grammars, limiting the words recognized in a given grammar and creating custom grammars is covered later.

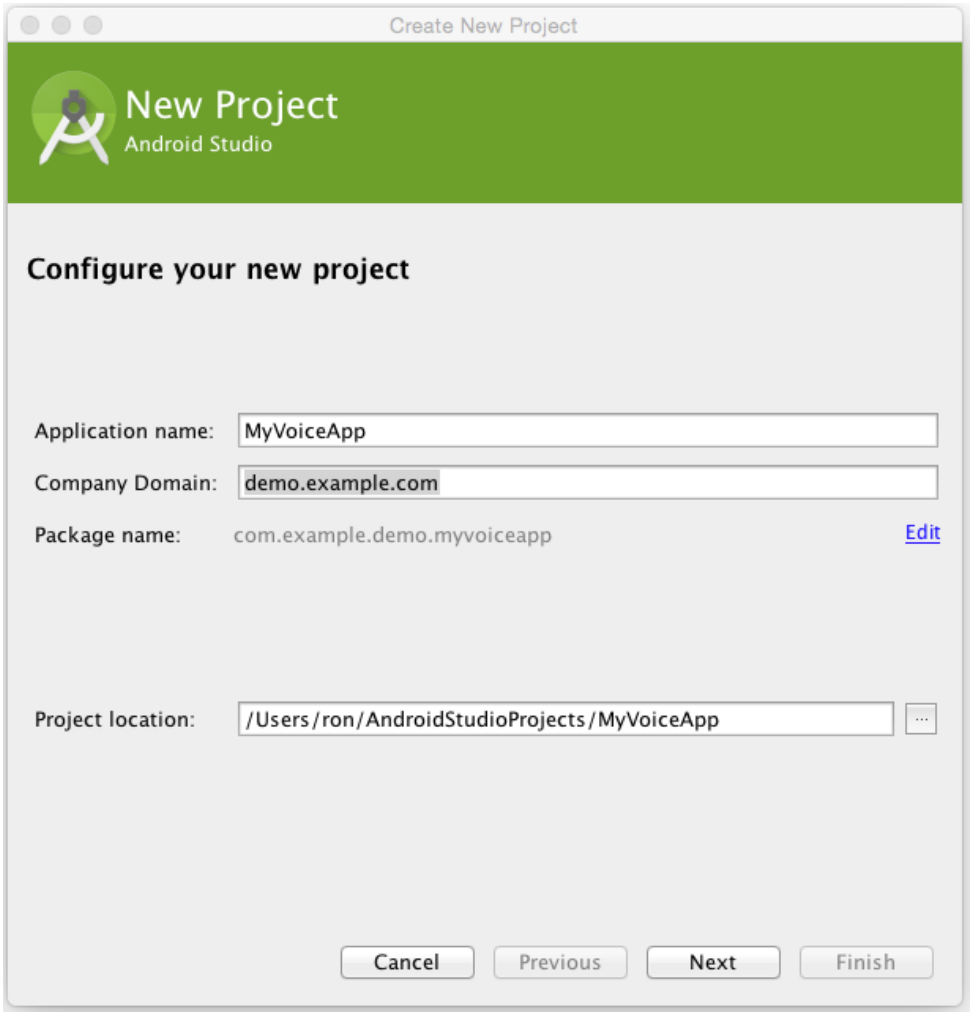
Building a Sample Voice Application for the M100

The following exercise will show you how to build a simple voice application using Android Studio and running it on your M100. The screen shots shown are taken from Android Studio running under MacOS X, but they should be accurate for Android Studio running under Linux or Windows as well.

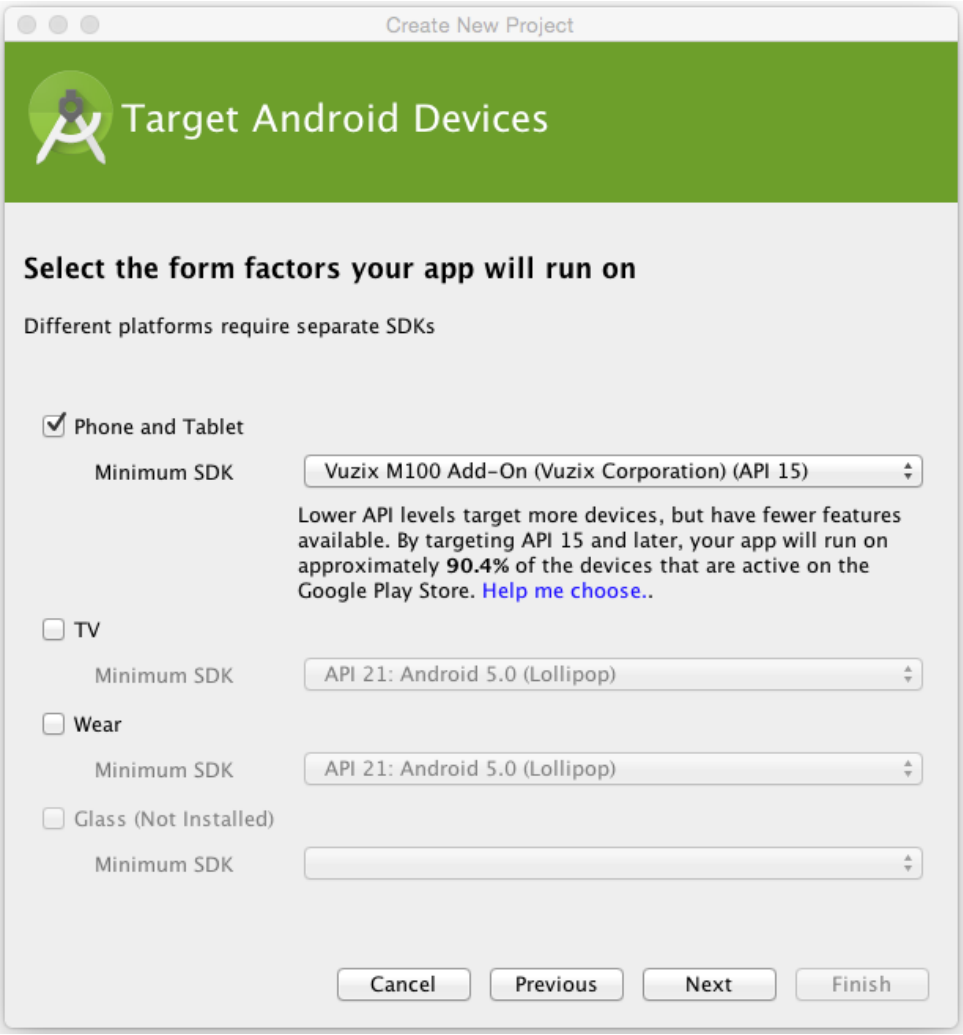
Begin by making sure you have Android Studio and the Vuzix M100 SDK installed. Then, launch Android Studio. You should see the following opening screen:



Click on the “Start a new Android Studio project” option and advance to the “Create New Project” window:

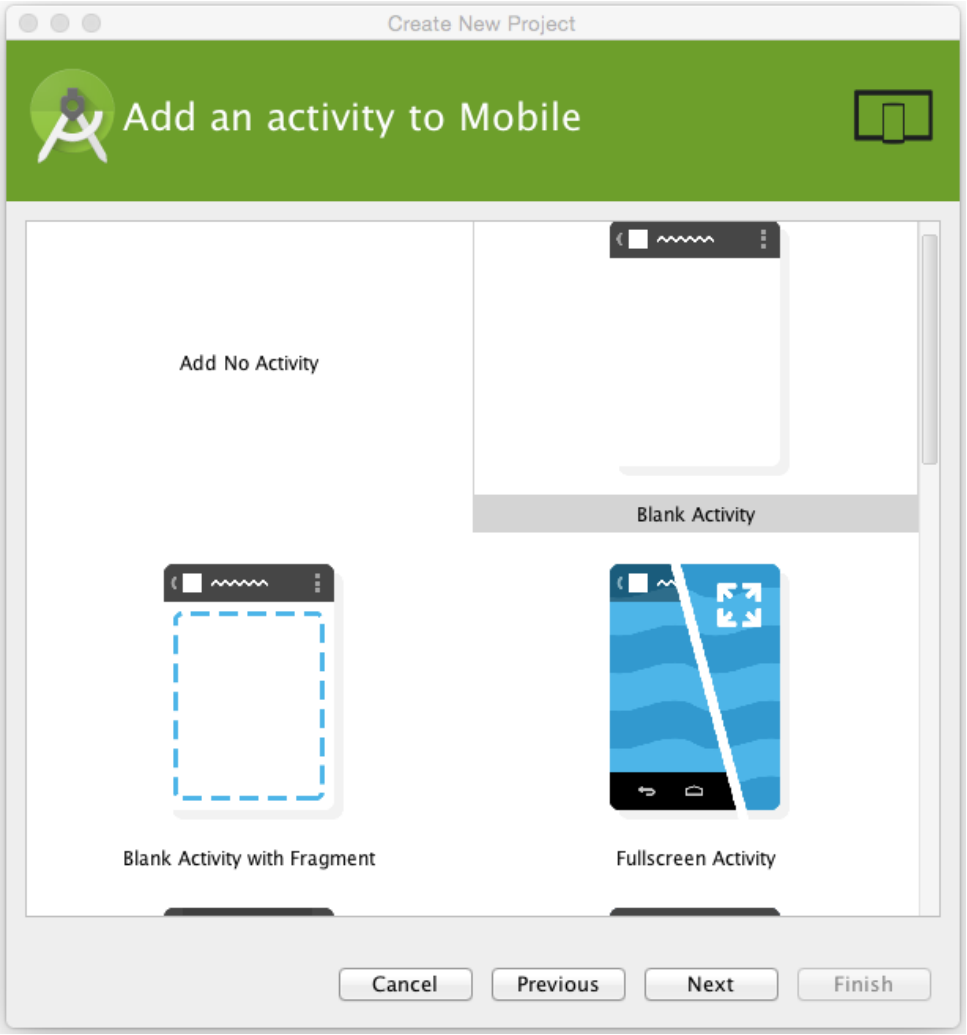


Choose a name for your project (such as "MyVoiceApp") and leave the "Company Domain" at whatever default value Android Studio chooses for you. You may change the Project location path if necessary. Then, click "Next":

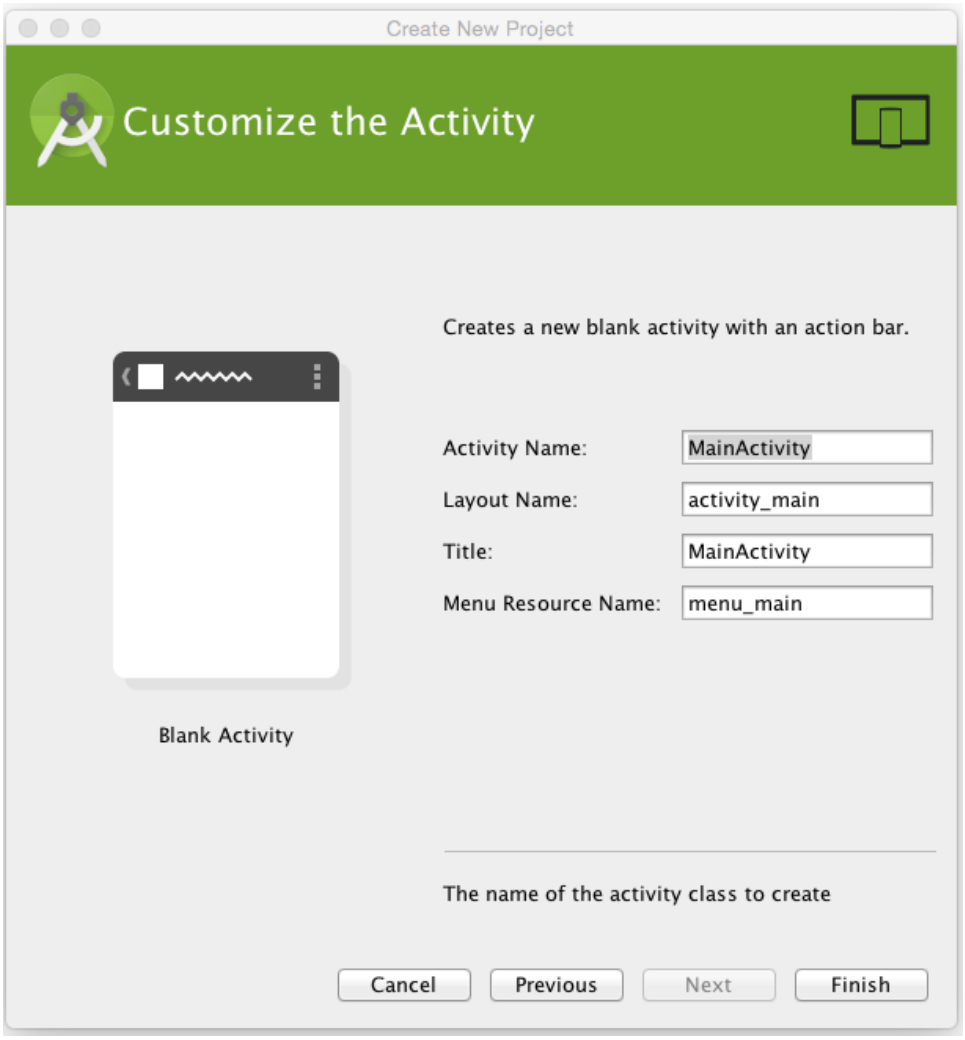


The key on this screen it to change the "Minimum SDK" from its default value to the "Vuzix M100 Add-On (Vuzix Corporation) (API 15)" option. If you do not see the "Vuzix M100 Add-On (Vuzix Corporation) (API 15)" option it means that you do not have the Vuzix SDK properly installed. Please review the Vuzix Developer documentation on installing Android Studio (and the SDK) and then try again.

Once you have chosen the proper minimum SDK, click on "Next". Assuming you are familiar with Android development, you will see the familiar screen for choosing the type of Activity you want to have in your application:

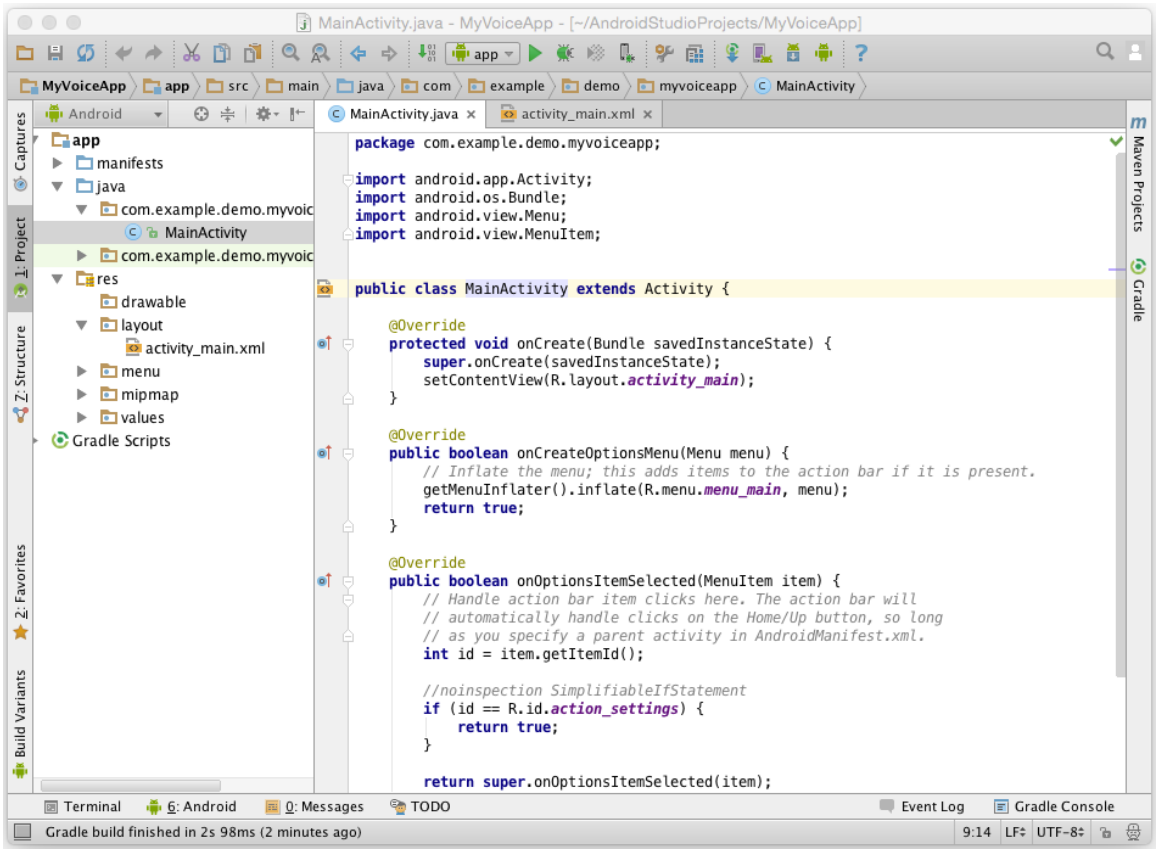


For the purposes of this demo, simply choose “Blank Activity” and then click “Next”. You will then be given the opportunity to customize details about this Activity:



It should be fine to leave all of these options at their default values. Click on "Finish" to complete the configuration of your project.

Once the project is configured by Android Studio, you will be presented with the main project window:



You may have to click on the “Project” tab to see this view if it does not appear by default. You may also need to navigate to the `MainActivity` class in the navigation panel in order to see the source code for `MainActivity` in the main project window.

Once you have the source code for `MainActivity` showing, you can begin to modify it in order to add voice control. Begin by modifying the import statements to include the classes necessary for implementing voice control . They should look like this:

```
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

import com.vuzix.speech.VoiceControl;
import android.widget.Toast;
```

Next, implement a custom `VoiceControl` class nested in the `MainActivity` class. We'll call the custom class `MyVoiceControl` and its implementation should look similar to this:

```
public class MyVoiceControl extends VoiceControl {
    public MyVoiceControl(Context context) {
        super(context);
    }

    protected void onRecognition(String result) {
        Toast.makeText(MainActivity.this,
                "Recognized: "+result,Toast.LENGTH_SHORT).show();
    }
}
```

This code will simply display a "short" Toast message (a small text message that appears as white text on a black background centered about 2/3 of the way down the device's screen) containing the word or words that were recognized.

Once `MyVoiceControl` is implemented, we can modify the default implementation of `MainActivity`'s `onCreate()` method to incorporate our custom voice control class and turn on the voice recognition mechanism.

```
private MyVoiceControl mVC;

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

    // Voice control code
    mVC = new MyVoiceControl(this);
    if (mVC != null)
    {
        mVC.on();
    }
}
```

In addition to overriding the `onCreate()` method of `MainActivity`, remember that we also need to turn the voice control mechanism on and off when our application is paused, resumed and ultimately destroyed:

```
@Override
public void onPause() {
    if (mVC != null) {
        mVC.off();
    }

    super.onPause();
}

@Override
public void onResume() {
    if (mVC != null) {
        mVC.on();
    }

    super.onResume();
}

@Override
public void onDestroy() {
    if (mVC != null) {
        mVC.off();
    }

    super.onDestroy();
}
```

Finally, we must also modify the `AndroidManifest.xml` file associated with our application to request the RECORD_AUDIO permission. This is accomplished by make use of a `uses-permission` tag just before the `application` section. The following snippet comes from the top of the `AndroidManifest.xml` file:

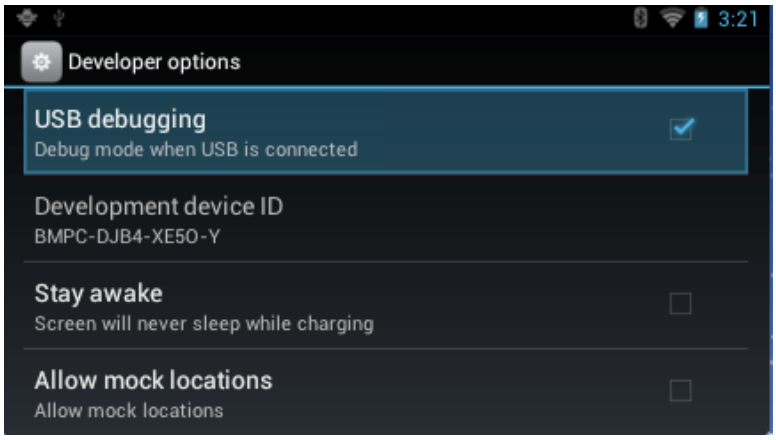
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.demo.myvoiceapp" >

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

    <application
        android:allowBackup="true"
```

At this point our sample application is complete and can be run on an M100. The standard procedure for running your code on an M100 is to attach the M100 via USB cable to your development machine and then choosing the "run" option in Android Studio. However before this will work you must make sure USB debugging is enabled on the M100.

Enabling USB Debugging on an M100

Before you can run a custom-developed application on an M100, you must make sure that the M100 has USB Debugging enabled. If it does not, Android Studio will not "see" the M100 and any attempt to run the code from Android Studio will result in Android Studio attempting to use an Android Simulator instead of the M100.

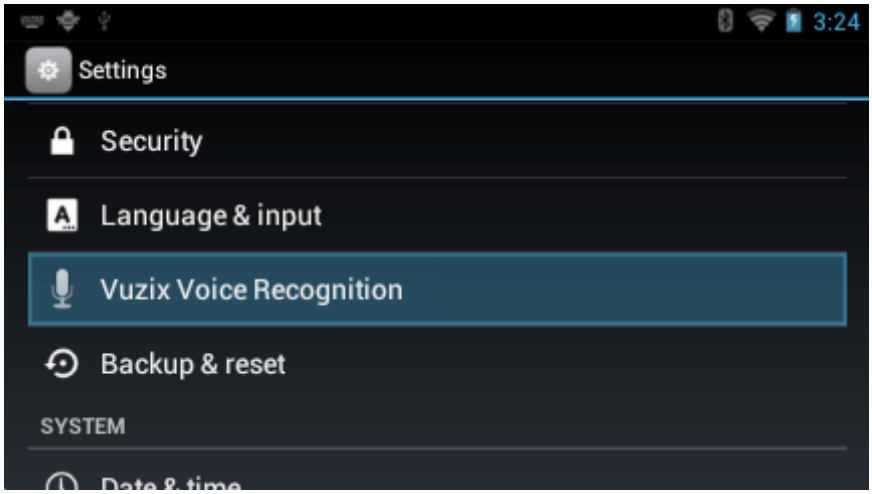
Enabling USB Debugging on the M100 is easy. Simply go to the "Settings" panel on the M100 and scroll all the way down to the "Developer Options" entry. Select this item to bring up the Developer options settings panel. The first entry in that panel is called "USB Debugging". Make sure the checkbox to the right of this item is checked:



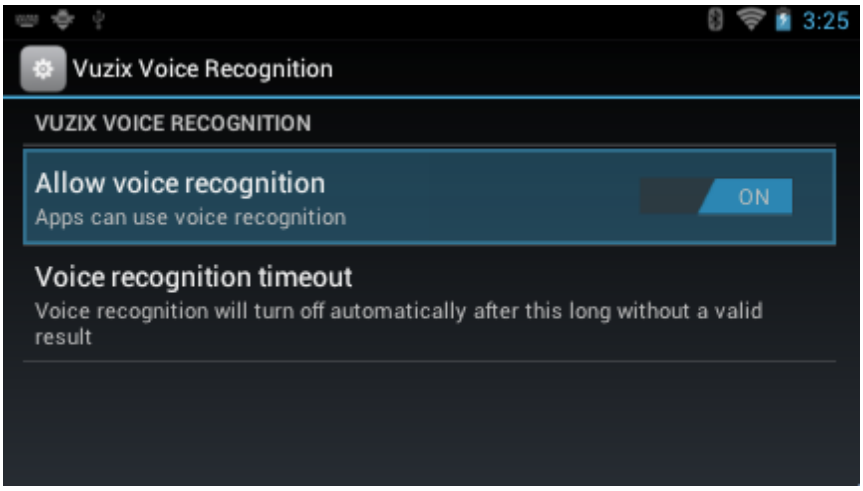
Once this item is checked, you will be able to run applications from Android Studio right on the M100. When running any application the utilizes voice recognition first make sure that voice recognition is enabled (see below).

Enabling Voice Recognition on an M100

Since voice recognition consumes a certain number of resources while running, the M100 does not have it enabled by default. While you can run a voice enabled application on the M100 even with voice recognition disabled (the app simply won't report any recognized phrases), you must make sure it is on if you'd like the app to process voice commands! To enabled voice recognition on the M100, find the "Vuzix Voice Recognition" settings panel in the Settings application:



Select this option and make sure that "Allow voice recognition" is turned "on":



Notice that there is a second option in the "Vuzix Voice Recognition" settings panel called "Vuzix recognition timeout". Feel free to explore this setting. It controls the amount of idle time the voice recognizer will wait before turning itself off. When the recognizer is off, the microphone icon at the top of the screen is grayed out (it is green when active). If the voice recognizer ever turns itself off while your app is running, you can usually restart it by speaking the phrase "voice on".

Using Additional Standard Grammars

The M100 Voice Control SDK comes with additional grammars that can be loaded at run time enabling your application to respond to a wider variety of commands. The M100 SDK comes with six predefined grammars that can be referenced using the following constants defined in the `com.vuzix.speech.Constants` class:

- `Constants.GRAMMAR_BASIC`
- `Constants.GRAMMAR_MEDIA`
- `Constants.GRAMMAR_CAMERA`
- `Constants.GRAMMAR_NAVIGATION`
- `Constants.GRAMMAR_WAREHOUSE`
- `Constants.GRAMMAR_MEDICAL`

When starting the voice recognition service the basic grammar is always pre-loaded. You can also load and unload any of these grammars by using available methods in the `VoiceControl` class.

For example, to enable our simple speech recognition example to recognize medical terms as well as the standard base grammar terms, we could modify the `onCreate()` method as follows:

```
import com.vuzix.speech.VoiceControl;
import com.vuzix.speech.Constants;

public class MainActivity extends Activity {
     private MyVoiceControl mVC;

     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);

          // Set up custom voice control class
          mVC = new MyVoiceControl(this);

          // Add MEDICAL grammar
          if (mVC != null) {
               mVC.addGrammar(Constants.MEDICAL);
          }

          // Make sure we were able to create mVC
          if (mVC == null) {
               Toast.makeText(this,"Can't create mVC",
                    Toast.LENGTH_SHORT).show();
               return;
          }

          // Turn voice control on
          mVC.on();

          // Rest of onCreate() code below…
     }
}
```

In addition to adding any of the predefined grammars, you can remove them as well (except for the base grammar). This is accomplished by using the `removeGrammar` method in a manner such as this:

```
          // Remove MEDICAL grammar
          if (mVC != null) {
               mVC.removeGrammar(Constants.MEDICAL);
          }
```

Using Additional Custom Grammars

While we strive to provide a broad set of grammars that can be used in multiple applications (and consider adding new "standard" grammars regularly), you may wish to develop a voice controlled application that requires commands not covered in any of the standard grammars. In these situations we do have a process in which we can create a custom grammar based on a set of words/command structure that you provide. There may be a fee for this service, please check with your Vuzix Developer Support representative and/or the Vuzix developer web site for more information.

When requesting a custom grammar from Vuzix, we will take the set of words/commands you wish to be able to recognize and return two files to you with the same base name and different extensions. For example, if we created a custom grammar for drawing and manipulating shapes on the screen, we might name the grammar "drawing_custom" and would produce two files with the following names:

- drawing_custom.bnf
- drawing_custom.lcf

Both files are important and should be stored in the `assets` directory in your code distribution. Loading the custom grammar is more complicated compared to the standard grammars as we must load the contents of the `.lcf` file into a byte array and then make use of an overloaded version of the `addGrammar()` API method.

To accomplish this in the context of our simple speech recognition example, we could add a private method to the MainActivity class to do the work:

```
private void loadCustomGrammar() {
     final android.content.res.Resources resource =
                    getResources();
     final android.content.res.AssetManager assets =
                    resources.getAssets();

     try {
          final java.io.InputStream fi =
           assets.open("drawing_custom.lcf");
```

```
            byte[] buf = new byte[fi.available()];
            while(fi.read(buf) > 0) {  }
            fi.close();

            mVC.addGrammar(buf,"Drawing Grammar");
      }
      catch(java.io.IOException ex) {
            ex.printStackTrace();
      }
}
```

Then we'd simply need to add a call to our `onCreate()` method in `MainActivity` to load the grammar, like so:

```
      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            // Set up custom voice control class
            mVC = new MyVoiceControl(this);

            if (mVC != null) {
                  loadCustomGrammar();
            }

            // rest of implementation here
      }
```

If you wish to remove a custom grammar, you must do so by using the String name you gave to the grammar when you loaded it. In our simple example above, the string name associated with the loaded grammar is "Drawing Grammar". To remove it, we'd do the following:

```
      if (mVC != null) {
            mVC.removeGrammar("Drawing Grammar");
      }
```

Gauging Accuracy of Recognized Speech

When using voice recognition you usually will have no problem when your voice commands are well enunciated by your users. However you may also notice that ambient noises and background conversation may sometimes register as words you are looking for. Even with the basic grammar, the M100 may hear someone say "mile" and have it register as "dial". How do you protect against these false positives?

The VoiceControl API contains a utility method named getConfScore() which is short for *Get Confidence Score*. The *confidence score* is a floating point number between 0 and 1. The higher the number, the more confident the speech recognition engine is that what it reported is in fact what was said. You can use this score to help filter out false positives. If, for example, you didn't want to consider any commands that didn't come with a 0.75 confidence score or higher, you could implement your `onRecognition` callback as follows:

```
protected void onRecognition(String result) {

     // Check confidence score, throw out anything
     // less than 0.75
     if (mVC.getConfScore() < 0.75)
          return;

     Toast.makeText(this,"Received: "+result,
               Toast.LENGTH_SHORT).show();
}
```

Restricting the Number of Words Recognized

Even within the context of a given grammar, there may be situations where you do not want to detect *all* the words in the grammar.

Suppose you wanted to write an application that only recognized the words "cut", "copy" and "paste". These are all defined in the base grammar, but so are a bunch of other words that you do *not* want to recognize. Or, suppose your application has a particular activity which, when viewed, only needs to respond to "cut", "copy" and "paste". In either case, you can limit the words recognized (out of the total number

of words recognizable across all loaded grammars) by providing a *word list*. A word list is simply an array of Java String objects which represent the exact words from the current loaded grammars that you would like to have recognized. This can be accomplished with the following code:

```
private void restrictWords() {
     // Create array of words to restrict to
     String wordlist[] = { "cut", "copy", "paste" };

     mVC.setWordlist(wordList);
}
```

When operating with a word list in place, you may find that your `onRecognition()` callback will still fire with an empty string as the argument when the speech recognizer detects a word that *isn't* on your word list.

To remove the word list restriction, simply call `setWordlist()` with a null parameter:

```
     // Remove previous word list restriction

     mVC.setWordlist(null);
```

Working with Subsets

Working with subsets
<< TO BE COMPLETED >>

Setting the Recognition Language

The underlying voice recognition technology used on the M100 uses the English language by default. It also has the theoretical capability of working with a host of other languages such as French, German, Russian, Polish, Japanese, Spanish and more!

In order to utilize other languages in the voice recognition system you must obtain and install a "language model" for the target language as well as any additional

grammars that have been compiled for that language. If the voice recognizer's language model is changed and you attempt to load a grammar that was compiled for a different language, the voice recognition process will fail.

For more information on how to obtain other language models and custom grammars to match other languages, please contact your Vuzix Developer Support representative.

The `VoiceControl` class uses the following API calls to manipulate the default recognition language:

- `setLanguage(int language)`
- `getCurrentLanguage()`
- `getPreviousLanguage()`

The language constants defined are as follows:

- `LANG_CANTONESE_HK`
- `LANG_CZECH`
- `LANG_DEFAULT`
- `LANG_DUTCH`
- `LANG_ENGLISH`
- `LANG_FRENCH`
- `LANG_GERMAN`
- `LANG_ITALIAN`
- `LANG_JAPANESE`
- `LANG_KOREAN`
- `LANG_MANDARIN_CN`
- `LANG_MANDARIN_TW`
- `LANG_POLISH`
- `LANG_PORTUGUESE`
- `LANG_RUSSIAN`
- `LANG_SPANISH`
- `LANG_SWEDISH`
- `LANG_TURKISH`

Getting Debug Information for Voice Control

When running voice enabled applications on the M100, especially applications you may be in the process of developing, it can sometimes be useful to see what is

happening at the SDK layer and below with respect to voice recognition. When running your application from Android Studio the logcat panel in your project window will usually display all entries that come from your application. This will include debug messages from the implementation of the `VoiceControl` class.

However, additional debug information from the underlying voice recognition engine can be seen if looking at the complete logcat data. This can be seen (along with a host of other data) by issuing the `adb logcat` command from your development machine.