Table of Contents

No Text While Driving

What You'll Learn

Getting Started

Design the Components

Add Behaviors to the Components

Programming an SMS Auto-Response
Storing the custom response in a database
Retrieving the Custom Response when the App Opens

Speaking the Incoming Texts Aloud
Adding Location Information to the Response
Sending the Location as part of the Response

The Complete App: No Text While Driving

Variations

Summary
No Text While Driving

This chapter walks you through the development of “No Text While Driving”, an app that auto-responds to SMS texts while you’re driving. The app, first created by a beginning computer science student with App Inventor, is similar to a now mass-produced app developed by State Farm Insurance. It is a great example of how App Inventor provides access to some of the great features of the Android phone, including SMS text processing, database management, text-to-speech, and the location sensor.
In January 2010, the National Safety Council (NSC) announced the results of a study which found that at least 28 percent of all traffic accidents—close to 1.6 million crashes every year—are caused by drivers using cell phones and texting [1]. Many states have banned cell phone use while driving altogether.

Daniel Finnegan was a student in a fall 2010 University of San Francisco App Inventor programming class. In brainstorming about a project for the class, Daniel thought about this driving and texting epidemic, and came up with an app that helps. The app he created, coined "No Text While Driving", responds to any arriving text with a response message such as, "I’m driving right now, I’ll text you back later." The app lets the user change the response for different situations—say if you’re going into a meeting or a movie.

When Daniel first introduced his idea, some in-class brainstorming resulted in a couple of great additions. First, it was observed that when you’re driving, the jingle of the texts coming in can kill you with curiosity—even if you know an app will auto-respond for you. Wouldn’t it be great if you could hear the texts spoken aloud?

The other idea was this: you’re a house husband (or wife) at home making dinner, and you’d like to know when your spouse will arrive from work. Of course you don’t want to endanger him or her by having them answer your call/text while driving, but you’d really like to know how far into that long commute they’ve progressed. The solution? Make use of the phone’s GPS capabilities and have “No Text While Driving” add the spouse’s current location in the response text as well.

We thought Daniel’s “No Text While Driving” idea was a good one so we developed it with all of these features as a tutorial on the App Inventor site. Some weeks later, State Farm Insurance created an Android app called “On the Move”. “On the Move” has strikingly similar functionality to “No Text While Driving”. The service is free to anyone as part of State Farm’s updated Pocket Agent® for Android™ application:
“It is our hope that this widget will prevent crashes and save lives,” said Laurette Stiles, Strategic Resources vice president at State Farm. “This new service will help drivers manage the temptation to read or respond to text messages when they are behind the wheel. We wanted to make this widget available free-of-charge as just one of the ways we’re working to keep our roadways safe for drivers.”[3].

We don’t know if Daniel’s app or the tutorial on the App Inventor site inspired, “No Text While Driving”, but it’s possible that it did. The interesting part of the story is that an app created in a beginning course, by a Creative Writing student, might have inspired this mass-produced piece of software, or at least contributed to the ecosystem that produced it! Its feasible because App Inventor has lowered the barriers of entry so that anyone with a good idea can quickly and inexpensively turn the idea into a tangible—well, virtual—reality, an interactive app.

In this chapter, you’ll learn how to build “No Text While Driving”, including the text-to-speech and location sensing capabilities. You’ll build it behavior by behavior, starting with the original auto-response. So open App Inventor and get ready! By the time you complete this app, you’ll have a pretty good idea how to program some of the most powerful functions on your Android device.

**What You'll Learn**

The tutorial covers the following components and concepts:
- The Texting component for sending texts and processing received texts.
- An input form for submitting the custom response message.
- The TinyDB database component for saving the customized message even after the app is closed.
- The Screen.Initialize event for loading the custom response when the app launches.
- The Text-to-Speech component for speaking the texts aloud
- The Location Sensor component for reporting the driver's current location

**Getting Started**

For this app to work, you'll need to load a text-to-speech module, Text-To-Speech Extended, on your phone. App Inventor and other apps rely on this module for the underlying voice synthesis. If you don't already have it, you can download it onto your phone with the Android Market. On your phone:

- Open the Market app
- Search for TTS
- Select the app *Text-To-Speech Extended* to install

Once the Text-To-Speech module is installed, open it to test its features. When it opens, set the default language as desired. Then select "Listen to Preview". If you don't hear anything, make sure the volume on your phone is turned on. You can also change the default Engine which can enhance the quality of the spoken words.

Connect to the App Inventor web site and start a new project. Name it "NoTextingWhileDriving" and also set the screen's Title to “NoTextingWhileDriving”. Open the Blocks Editor and connect to the phone.

**Design the Components**

The user interface for NoTextingWhileDriving is simple: it has a label that displays the current response along with a text box and a button for submitting a change. You'll also need to drag in a Texting component, a TinyDB component, a Text-to-speech component, and a Location Sensor component, all of which will appear in the "non-visual" component area. You can build the components from the snapshot of the Component Designer below, or the table of instructions that follows:
You can build the components shown above by dragging out the following components:

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Palette Group</th>
<th>What you'll name it</th>
<th>Purpose of Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Basic</td>
<td>PromptLabel</td>
<td>Let the user know how the app works</td>
</tr>
<tr>
<td>Label</td>
<td>Basic</td>
<td>ResponseLabel</td>
<td>The response that will be sent back to the sender of original text</td>
</tr>
<tr>
<td>TextBox</td>
<td>Basic</td>
<td>NewResponseTextbox</td>
<td>User will enter custom response here.</td>
</tr>
<tr>
<td>Button</td>
<td>Basic</td>
<td>SubmitResponseButton</td>
<td>User clicks this to submit response</td>
</tr>
<tr>
<td>Texting</td>
<td>Social</td>
<td>Texting1</td>
<td>The component that processes the texts</td>
</tr>
<tr>
<td>TinyDB</td>
<td>Basic</td>
<td>TinyDB1</td>
<td>The component that will store the response in the database</td>
</tr>
<tr>
<td>TextToSpeech</td>
<td>Other stuff</td>
<td>TextToSpeech1</td>
<td>The component that will speak the texts aloud</td>
</tr>
<tr>
<td>LocationSensor</td>
<td>Sensors</td>
<td>LocationSensor1</td>
<td>The component that knows where the phone is</td>
</tr>
</tbody>
</table>

Set the properties of the components in the following way:

- Set the Text of PromptLabel to "The text below will be sent in response to all SMS texts received while this app is running."
- Set the Text of ResponseLabel to "I'm driving right now, I'll contact you shortly."
- Set the Text of NewResponseTextbox to "".
- Set the Hint of NewResponseTextbox to "enter new response text".
- Set the Text of SubmitResponseButton to "Modify Response".
Add Behaviors to the Components
You’ll start by programming the basic auto-response behavior, then successively adding more functionality.

Programming an SMS Auto-Response
For the auto-response behavior, you’ll use App Inventor’s Texting component. You can think of this component as a little person inside your phone that knows how to read and write texts. For reading texts, the component provides a “when Texting.MessageReceived” event block. You can drag this block out and place blocks inside it for whatever you want to see happen when a text is received.

To program the response text, you’ll place a Texting.SendMessage block within the MessageReceived block. Texting.SendMessage actually sends the text—so you’ll first need to tell the component the message to send, and who to send it to, before calling SendMessage.

You’ll need the following blocks for this auto-response behavior:

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Drawer</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texting1.MessageReceived</td>
<td>Texting1</td>
<td>the event-handler that is triggered when phone received a text</td>
</tr>
<tr>
<td>set Texting1.PhoneNumber to</td>
<td>Texting1</td>
<td>set the PhoneNumber property before sending</td>
</tr>
<tr>
<td>call number</td>
<td>My Definitions</td>
<td>this is the phone number of the person who sent the text</td>
</tr>
<tr>
<td>set Texting1.Message to</td>
<td>Texting1</td>
<td>set the Message property before sending</td>
</tr>
<tr>
<td>MessageText.Text</td>
<td>MessageText</td>
<td>This is the message the user has entered.</td>
</tr>
<tr>
<td>Texting1.SendMessage</td>
<td>Texting</td>
<td>send the message</td>
</tr>
</tbody>
</table>

The blocks should look like this:

If you’re following along building this app, note that the yellow boxes are comments—you can add them by right-clicking on a block and selecting “Add Comment”. You don’t have to add these-- comments are for decoration only and don’t cause the app to behave differently. Comments are important, however, both for you as you build the app and modify it later, and
for others who might customize it. The one thing everyone agrees on about software is that it changes and transforms often. For this reason, commenting code is very important in software engineering, and especially so with open-source software.

**How the Blocks Work**
When the phone receives any text message, the Texting1.MessageReceived event is triggered. The phone number of the sender is in the argument `number`, and the message received is in the argument `messageText`. In response, the app sends a text message to the sender. To send a text, the app first sets the two key properties of the Texting component: PhoneNumber and Message. Texting.PhoneNumber is set to the number of the sender, and Texting.Message is set to the text you typed into ResponseLabel--"I'm driving right now, I'll contact you shortly." Once these are set, the app calls SendMessage to actually send the response.

| Test the behavior. You'll need a second phone to test this behavior. From the second phone, send a text to the phone that is running the app. Does the second phone receive the response text? |

**Storing the custom response in a database**
Next, modify the app so it stores the custom response message the user enters. When the user enters a text and clicks the Modify Response button, the app should 1) move the text the user has entered from the textbox NewResponseText into the ResponseLabel, and 2) store the text in the phone’s database.

The response text needs to be stored in the database because the user expects that data to be persistent. Persistent data is like the information you enter when you modify your Facebook profile—if you change your status there, you expect that newly entered status to be the same when you leave Facebook and come back to it later. In this case, when the user closes the “No Texting While Driving” app, then re-opens it later, he will expect the last response message he entered to be remembered by the system.

Data stored in component properties, such as the ResponseLabel.Text property, is transient data. Transient data is like your short-term memory, and the phone “forgets” it as soon as an app closes. If you want to remember something long-term, you have to transfer it from short-term memory (a component property or variable) to long-term memory (a database).

You can store data persistently using the TinyDB component. TinyDB provides two blocks: StoreValue and GetValue. The former allows the app to store a tagged piece of information into the phone’s database, while the latter lets the app retrieve one. Here are the blocks you’ll need for transferring a newly entered response into the ResponseLabel and storing it in the database:
The blocks should look like this:

How the Blocks Work
Think of the input forms you use—you first enter something in a textbox then click a submit button to tell the system to process it. The input form for this app is no different. When the user clicks the SubmitResponseButton, the SubmitResponseButton.Click event is triggered. The event-handler first copies (sets) what the user has entered in NewResponseTextBox into the ResponseLabel. Recall that ResponseLabel holds the message that will be sent out in auto-response, so it is imperative that the new custom message is placed there.

Next, the app uses TinyDB to store the text it just put in ResponseLabel into the database. The text "responseMessage" is used as a tag to uniquely identify the information—later, you'll use that same tag to retrieve the message from the database.

Retrieving the Custom Response when the App Opens
The reason for storing the custom response in the database is so that, when the user closes then later reopens the app, that custom response can be loaded back into the app.

App Inventor provides a special event block that is triggered when the app opens: Screen1.Initialize. If you drag this event block out and place blocks in it, those blocks will be executed right when the app begins.

For this app, your Screen1.Initialize event-handler should check to see if a custom response
has been put in the database. If it has, it should be loaded in using TinyDB’s GetValue function. Here are the blocks you’ll need:

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Drawer</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>def variable</td>
<td></td>
<td>A temporary variable to hold the retrieved data</td>
</tr>
<tr>
<td>text (blank)</td>
<td>Text</td>
<td>Initial value for the variable can be anything</td>
</tr>
<tr>
<td>Screen1.Initialize</td>
<td>Screen1</td>
<td>This is triggered when app begins</td>
</tr>
<tr>
<td>set response to</td>
<td>My Definitions</td>
<td>you’ll set the variable to the value retrieved from db</td>
</tr>
<tr>
<td>TinyDB1.GetValue</td>
<td>TinyDB</td>
<td>get the stored response text from the database</td>
</tr>
<tr>
<td>text (&quot;responseMessage&quot;)</td>
<td>Text</td>
<td>plug into tag slot of GetValue, make sure text is same as was used in StoreValue above</td>
</tr>
<tr>
<td>if test</td>
<td></td>
<td>to ask if the retrieved value has some text</td>
</tr>
<tr>
<td>&gt; block</td>
<td>Math</td>
<td>check if length of retrieved value is greater than (&gt;) 0</td>
</tr>
<tr>
<td>length text</td>
<td>Text</td>
<td>check if length of retrieved value is greater than 0</td>
</tr>
<tr>
<td>global response</td>
<td>My Definitions</td>
<td>this variable holds the value retrieved from GetValue</td>
</tr>
<tr>
<td>number (0)</td>
<td>Math</td>
<td>to compare with length of response</td>
</tr>
<tr>
<td>set MessageTextbox.Text to</td>
<td>MessageTextbox</td>
<td>if we retrieved something, place it in MessageTextbox</td>
</tr>
<tr>
<td>global response</td>
<td>My Definitions</td>
<td>this variable holds the value retrieved from GetValue</td>
</tr>
</tbody>
</table>

This is how the blocks should look:

**How the Blocks Work**

When the app begins, the Screen1.Initialize event is triggered. The app calls the TinyDB1.GetValue with a tag of "responseMessage"-- the same tag used when you stored the user's entry earlier. The resulting value is placed in the variable response.

The variable response is used so that the value returned from the database can be checked. If it has a length of 0, then there was no database entry with a tag of "responseMessage"-- something that will occur the first time a user runs this app. But if the length is greater than 0, the app knows that a custom response has been stored previously, and the retrieved value can
be placed in ResponseLabel which the user will see and which is used as the message for any response texts sent.

**Test the behavior.** Enter a new response message in the NewResponseTextBox and click the SubmitResponseButton. Then restart the app by clicking the Restart App button in the Blocks Editor. This will cause the app to re-initialize just like it will when a user closes the app and reopens it later. Does the custom message you entered appear?

**Speaking the Incoming Texts Aloud**

In this section, you’ll modify the app so that, when a text is received, the sender’s phone number, along with the message, is spoken aloud. The idea here is that when you’re driving and here a text come in, you might be tempted to check the text even if you know the app is sending an auto-response. With Text-to-speech, you can hear the incoming texts and keep your hands on the wheel.

Android devices provide text-to-speech capabilities and App Inventor provides a component, TextToSpeech, which will speak out any text you give it. Note that here “text” is meant in the general sense of the word—a sequence of letters, digits and punctuation—and not just an SMS text.

In the Getting Started section of this app you were instructed to download a text-to- speech module from the Android market. If you didn't do so, you should do so now. Once the Text-To-Speech module is installed and configured as desired, you can use the TextToSpeech component within App Inventor.

The TextToSpeech component is very simple to use—you just call its Speak function and plug in the text you want spoken into its message slot. For instance, the following function call would speak, “Hello World”:

For the “No Texting While Driving app”, you’ll need to provide a more complicated message to be spoken, one that includes both message received and the phone of who sent it. Instead of plugging in a static text object like the red “Hello World” block, you’ll plug in a make text block. make text allows you to combine multiple pieces of information into a composite text object. You’ll need to make the call to TextToSpeech.Speak within the Texting.MessageReceived event handler you programmed earlier. The blocks you programmed earlier handle this event by setting the PhoneNumber and Message properties of the Texting component appropriately and then sending the response text.

You’ll extend that event handler by adding the following blocks:
The Texting1.MessageReceived event-handler will now look like this:

How the Blocks Work
After the response is sent, the TextToSpeech1.Speak function is called. You can plug any text object into the message slot of the Speak function. In this case, make text is used to build the words to be spoken. make text concatenates together the text "SMS text received from", the phone number from which the message was received (number), the text ".The Message is:", and finally the message received (messageText). If the text "hello" was received from the number "111-2222", the words "SMS text received from 111-2222. The message is hello." would be spoken.

Test the behavior. You'll need a second phone to test the behavior. From the second phone, send a text to the phone that is running the app. Does the phone running the app speak the text aloud? Does it still send an automated response?

Adding Location Information to the Response
App’s like Facebook’s Place and Google’s Latitude use GPS information to help people track each other’s location. There are major privacy concerns with such apps, one
reason being that location tracking kindles people’s fear of a “Big Brother” apparatus that a totalitarian government might set up to track its citizen’s whereabouts. Of course there are more personal concerns as well—many people are happy keeping there whereabouts private unless someone asks them where they are and they choose to answer.

Of course location information can be quite useful as well—one thinks of a lost child or hikers lost in the woods. Within the context of “No Text While Driving”, the driver’s location information can be used to convey a bit more information in response to incoming texts. Instead of just sending back, “I’m driving”, the response can be something like “I’m driving and I’m at 3413 Cherry Avenue”. For someone awaiting the arrival of a friend or loved one, this extra information can allow them to plan accordingly.

App Inventor provides the LocationSensor component for interfacing with the GPS functionality of the phone. GPS stands for Geographical Positioning System.[UI2] Besides latitude and longitude information, the LocationSensor also will provide the current street address, making use of Google Maps information to do so.

The LocationSensor doesn’t always have a reading. For this reason, care needs to be taken to use the component properly. Specifically, your app should respond to the LocationSensor.LocationChanged event handler. A LocationChanged.event occurs when the phone’s location sensor first gets a reading, and when the phone is moved to produce a new reading. The scheme we will use for this app is to respond to the LocationChanged event by placing the current address in a variable lastKnownLocation. Later, we’ll change the response message to incorporate this address.

Here are the blocks you’ll need:

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Drawer</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>def variable</td>
<td>Definitions</td>
<td>create a variable to hold the last read address</td>
</tr>
<tr>
<td>text</td>
<td>Text</td>
<td>set the default value in case the phone’s sensor is not working</td>
</tr>
<tr>
<td>LocationSensor1.LocationChanged</td>
<td>LocationSensor1</td>
<td>this is triggered on the first location reading and every change.</td>
</tr>
<tr>
<td>set lastKnownLocation to</td>
<td>My Definitions</td>
<td>set this variable to be used later.</td>
</tr>
<tr>
<td>LocationSensor1.CurrentAddress</td>
<td>LocationSensor1</td>
<td>this is a street address such as &quot;2222 Willet Street, Atlanta, Georgia&quot;</td>
</tr>
</tbody>
</table>

Here’s how the blocks should look:
How the Blocks Work
The LocationSensor1.LocationChanged event is triggered the first time the sensor gets a location reading, and when the device is moved so that a new reading is performed. Since you eventually want to send a street address as part of the response message, the LocationSensor’s CurrentAddress function is called to get that information. Behind the scenes, this function makes a call to the Google Maps API in order to determine the closest street address for the latitude and longitude that the sensor read.

Note that with these blocks you’ve only finished half of the job. The app still needs to incorporate the location information, now stored in the variable lastKnownLocation, in the response SMS text sent back to the sender.

Sending the Location as part of the Response
Using the variable lastKnownLocation, you can modify the Texting1.MessageReceived event handler to add location information to the response.

You’ll need the following blocks:

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Drawer</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>make text</td>
<td>Text</td>
<td>if there is a location reading, build a compound text object</td>
</tr>
<tr>
<td>ResponseLabel.Text</td>
<td>MessageTextBox</td>
<td>this is the (custom) message in the text box</td>
</tr>
<tr>
<td>text (“My last known location is:”)</td>
<td>Text</td>
<td>This will be spoken after the custom message (note the leading space)</td>
</tr>
<tr>
<td>global lastKnownLocation</td>
<td>LocationSensor</td>
<td>This is an address such as “2222 Willard Street, Atlanta, Georgia”</td>
</tr>
</tbody>
</table>

Here is how the updated event-handler should look:
**How the Blocks Work**

This behavior works in consort with the LocationSensor1.LocationChanged event and the variable lastKnownLocation. Instead of directly sending a message of the text in ResponseLabel.Text, the app first builds a message using make text. It combines the response text in ResponseLabel.Text with the text "My location is " and then the variable lastKnownLocation. The default value of the variable lastKnownLocation is “unknown”, so if there hasn’t yet been a reading by the location sensor, the second part of response message will contain the text “My last known location is: unknown”. If there has been a reading, the second part of the response will be something like “My last known location is: 876 Willard Street, San Francisco, CA. 95422”

**Test the behavior.** From the second phone, send a text to the phone that is running the app. Does the second phone receive the response text? Does it include location information?

**The Complete App: No Text While Driving**
Variations

Once you get the app working, you might want to explore some variations. For example,

- Write a version that lets the user define custom responses for particular incoming phone numbers.
- Write a version that sends custom responses based on the whether the user is within certain latitude/longitude boundaries. So if the app determines you're within a particular meeting room, it will send back “Bob is in the meeting room and can't text right now.”
- Write a version that sounds an alarm when a text is received from a number in a "notify" list.

Summary

Here are some of the concepts covered in this tutorial:
• The Texting component can be used both to send text messages and process the ones that are received. Before calling SendMessage, you should set the PhoneNumber and Message properties of the Texting component. To respond to an incoming text, program the Texting.MessageReceived handler.
• The TinyDB component is used to store information persistently, in the phone’s database, so that the data can be re-loaded each time the app is opened.
• The TextToSpeech component takes any text object and speaks it aloud. It relies on an Android module, Text-To-Speech Extended, which must be downloaded to the phone for the app to work.
• make text is used to piece together (concatenate) separate text items into a single text object.
• The LocationSensor component can report the phone’s latitude, longitude, and current street address. To ensure that it has a reading, you should access its data within the LocationChanged event-handler, which is triggered the first time a reading is made and on every change thereafter.