|  |  |
| --- | --- |
| This tutorial uses the App Inventor Texting Component to receive and send text messages to a list of registered members. Messages sent to the hub by registered members are distributed to all the members of the hub. This version is a variation of the [MIT tutorial](http://beta.appinventor.mit.edu/learn/tutorials/broadcasterhub/broadcasterhub.html). It assumes that you know how to set up your device for using the [Texting component over Wifi](http://appinventor.mit.edu/explore/content/google-voice.html) (i.e., for devices with no Sim card or mobile service plan). **IMPORTANT NOTE: Google Voice has been updated and is no longer working for use with the texting component. In order to test apps that require the texting component, disable Google Voice in App Inventor and use an Android device that has cellular service (e.g. an Android cellphone).**  The lesson includes several programming exercises that add enhancements to the basic app.  **Objectives:** In this lesson you will learn to :   * create an app that   + uses the Texting component and Google Voice to create a hub for group SMS messages,   + follows a complex *if-else* algorithm for processing incoming texts; * gain additional experience using procedures to organize an app.   **NOTE: To test this app, it is necessary to package and install it on your device.** | ***[Click here to watch the preview](https://www.youtube.com/watch?v=8hGirBMNo-4)*** |

## IMPORTANT NOTE: Google Voice has been updated and is no longer working for use with the texting component. In order to test apps that require the texting component, disable Google Voice in App Inventor and use an Android device that has cellular service (e.g. an Android cellphone).

# 

# Getting Ready

## To get started, [open App Inventor with the BroadcastHub Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit7/templates/BroadcastHub/BroadcastHubTemplate.asc) in a separate tab and follow along with these tutorials.

## 

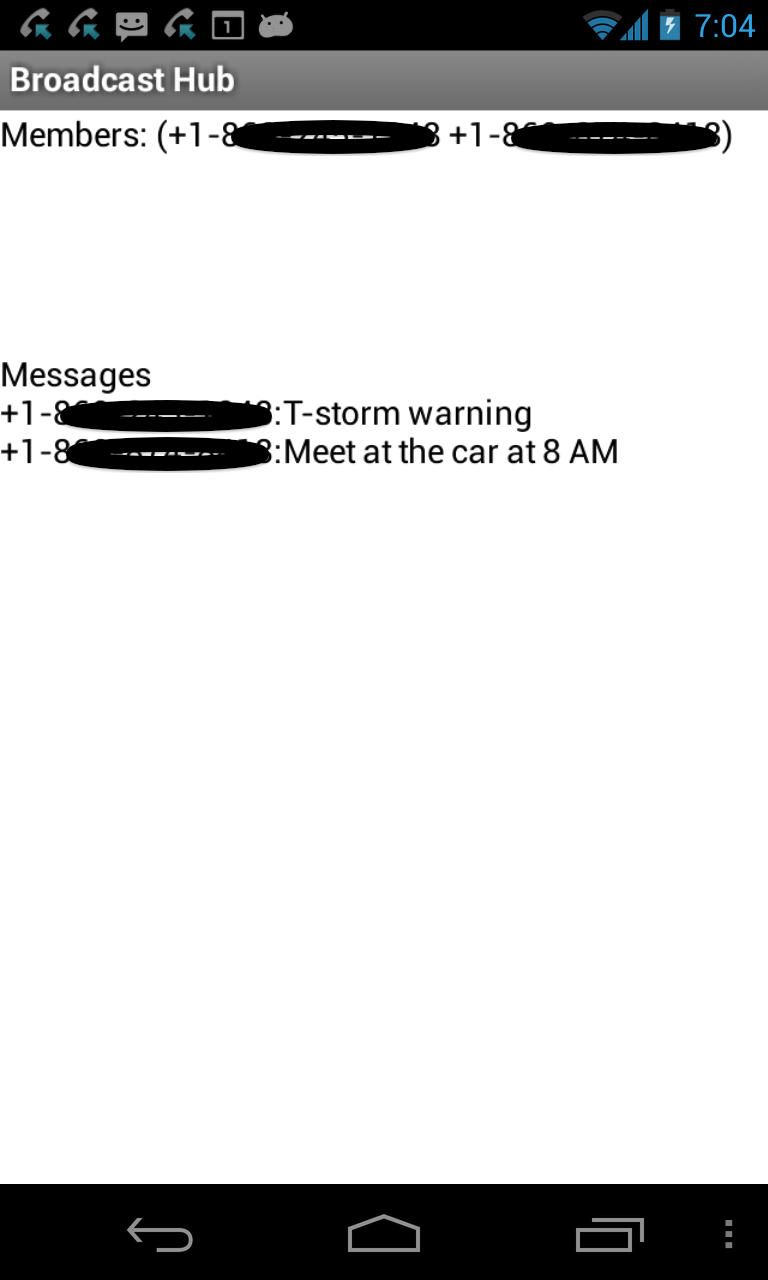
# Broadcast Hub Tutorial

## Basic Idea

This tutorial uses the App Inventor Texting Component to receive and send text messages to a list of registered members. This version is a variation of the [MIT tutorial](http://beta.appinventor.mit.edu/learn/tutorials/broadcasterhub/broadcasterhub.html). It assumes that you know how to set up your device for using the [Texting component over Wifi](http://appinventor.mit.edu/explore/content/google-voice.html) (i.e., for devices with no Sim card or mobile service plan). **IMPORTANT NOTE: Google Voice has been updated and is no longer working for use with the texting component. In order to test apps that require the texting component, disable Google Voice in App Inventor and use an Android device that has cellular service (e.g. an Android cellphone).**

Here’s how it works: The phone or device running this app serves as a hub. Other cell phone users register to join the hub by sending a ‘joinhub’ message to the app. When the app receives such a message, it adds the sender’s phone number to a list. Then whenever a message is received at the hub from a member of the broadcast list, the app sends the message to every user on the list. This tutorial describes the app’s design and implementation and contains several suggestions for enhancements.

## Broadcast Hub UI

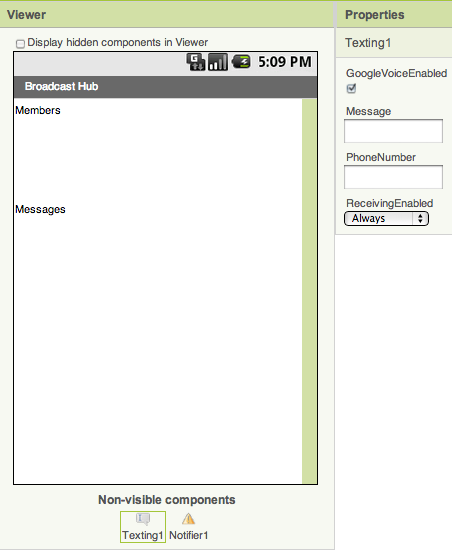
**The User Interface**

1. The very simple UI consists of two *Labels* that are used to display information. The *members label* displays the list of the phones currently on the broadcast list. The *messages label* displays the messages received and the phone number of who sent the message.
2. ***Non-visible Component 1:*** A *Texting* component is used to receive and send messages.
3. ***Non-visible Component 2:*** A *Notifier* is used to display a message when a message is received from a phone that is not a member of the broadcast list.

## 

## 

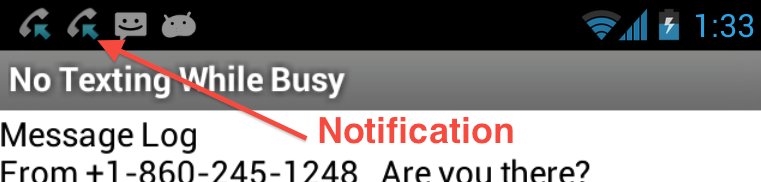
## The Texting Component

[NOTE: This description of the Texting component was covered in the *NoTextWhileBusy* tutorial. If you’ve done that lesson, you can skip this.] 

The *Texting* component is a non-visible component that is used to send and receive text messages. As you can see here it has four properties:

* **GoogleVoiceEnabled**: If you want to use the component over Wifi, you must have it enabled for Google Voice. Details are provided below on how to set it for this usage.
* **Message**: A default message may be specified or left blank.
* **PhoneNumber**: Messages will be sent to this default phone number, which may be left blank.
* **ReceivingEnabled:** This property controls how the phone responds to incoming (received) messages. There are three possible settings:

1. **ReceivingEnabled: OFF.** This corresponds to the numerical value 1. When set to ‘Off’, received messages will be ignored by the app.
2. **ReceivingEnabled: Foreground.** This corresponds to the numerical value 2. When set to ‘Foreground’, the Texting component will receive messages only when it is running in the foreground -- i.e., when it is visible on the phone. This behavior corresponds to the original version of the Texting component.
3. **ReceivingEnabled: Always.** This corresponds to numerical value 3. When set to ‘Always’, the Texting component will receive messages when it is running in the foreground, but also when it is not running. An app that uses a Texting component set to ‘Always’ can receive messages as long as it is installed on the phone. Its behavior in this case is to put a *notification* in the device’s status bar:

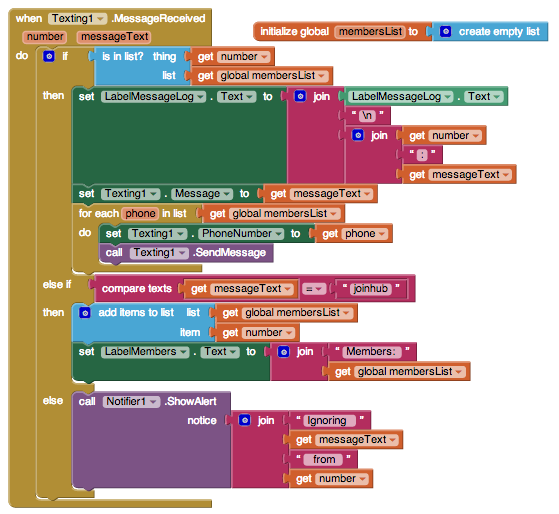
****

When the user pulls down the status bar and taps on the notification, the app will start up and appear in the foreground. We’ll try this version after we build the app. For now, make sure you have the Texting component’s properties set as follows:

* + GoogleVoiceEnabled: True **IMPORTANT UPDATE: Set this property to FALSE - Google Voice has been updated and is no longer working for use with the texting component. In order to test apps that require the texting component, disable Google Voice in App Inventor and use an Android device that has cellular service (e.g. an Android cellphone).**
  + ReceivingEnabled: Always

## The Blocks

This app contains a global variable, *membersList*, and one main block, which handles all its behavior:



As in the *NoTextingWhileBusy* tutorial, the *MessageReceived* block is used to process incoming text messages. It has two properties -- the *number* that sent the message and the *messageText* that was received from that number. This block will file only if *Texting*.*ReceivingEnabled* property is set to *Foreground* and the app is running in the foreground or if it set to *Always* and the app has been installed on the phone.

### 

### Broadcast Hub Algorithm

The app uses an ***if-else*** algorithm with an ***embedded for-each*** loop. Here is a *pseudocode* version:

**if** the message received is from a number already on the memberList

do: Append the message to the message log

Set it as the value of the Texting.message property

***For each phone number on the membersList***

***Send the message to that phone number***

**else if** the message received is ‘joinhub’

do: Add the sender’s phone number to the membersList

Display the list of phone numbers in the UI

**else**

do: Notify the user that a message has been received from a non-member

The *if-els*estatement has three mutually exclusive cases (or branches). For each message received, only one of the following conditions will be true and only one of the associated actions will be done:

* Condition: The incoming phone number is on the list:

Action: Send everyone the incoming message

* Condition: The incoming message is ‘joinhub’:

Action: Add the phone number to the list

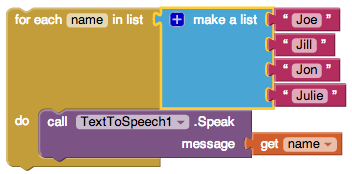
* Condition: None of the above:

Action: Notify the user that a non-member message has been received

The *for-each* loop is a special loop block that is meant to be used with lists.



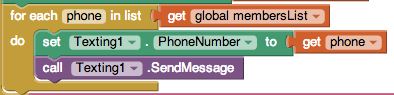
It has a built-in variable *i* (which in our code we have re-named *phone*) and a slot where you must insert a *list*. You can insert either a variable -- but make sure the variable you insert there actually contains a list -- or a *make a list* block. The variable *i* will be set to each item in the corresponding list. Here’s an example of a for-each loop that will *speak* each *name* on a list of names:



As you can see in this example, it’s a good idea to rename *i* to be descriptive of the items in the list -- in this case *name.*

What happens here: On the first *iteration* of the loop, the variable *name* will be set to “Joe” and the app will speak ‘Joe’. On the second iteration, *name* will be set to “Jill” and that name will be spoken. And so on until each member of the list has been processed. In this example we say that the loop will make *4 iterations* -- 4 cycles or 4 passes through the loop -- one for each item in the list of names.

For the Broadcast hub, the for-each loop sets the *phone* variable to each item in the *membersList* and then sends a text message to that phone.



In this case the number of iterations depends on how many members have registered for the hub.

## 

## 

# Setting Up for a Device with Mobile Service

If you are using a device that has mobile service, then no special setup is required. You can use this app, with *GoogleVoiceEnabled* set to false.

# Setting Up to Use Texting over the Wifi/Internet

If you are using a tablet or a phone that does not have mobile service you can use this app if you following [these setup instructions](http://appinventor.mit.edu/explore/content/google-voice.html).

# Running and Testing the App

**To test this app, it is necessary to package and install it on your device.** See the [How To: Package and Install Your app](https://docs.google.com/document/d/1l5jAubqxUZEKbIOPIuGYk26XnxxblDkCidxwRPAuQCM/edit?usp=sharing) for an example of how to package and install this app on your device. You’ll also need more than one device to test it properly:

* Use two or three texting devices, only one of which needs to be an Android device. Install the app on an Android device, making it the broadcast hub. Have the other devices send the ‘joinhub’ message to the broadcast hub. Once several devices have joined the hub, have one of them send a message to the hub. It should be distributed to all the devices.

This is a tricky app to test and debug because there are lots of things that could go wrong to prevent the app from working properly, even if its code is correct. Here are some things to think about:

* Make sure all the devices have proper Wifi connections if you are using Wifi, rather than mobile service.
* Make sure the broadcast hub device has GoogleVoice installed and enabled and the user has a Google Voice account. See these [setup instructions](http://appinventor.mit.edu/explore/content/google-voice.html). **IMPORTANT NOTE: Google Voice has been updated and is no longer working for use with the texting component. In order to test apps that require the texting component, disable Google Voice in App Inventor and use an Android device that has cellular service (e.g. an Android cellphone).**
* Remember: For each change you make to the app, it will be necessary to [re-package it and re-install it on the device.](https://docs.google.com/document/d/1l5jAubqxUZEKbIOPIuGYk26XnxxblDkCidxwRPAuQCM/edit?usp=sharing)

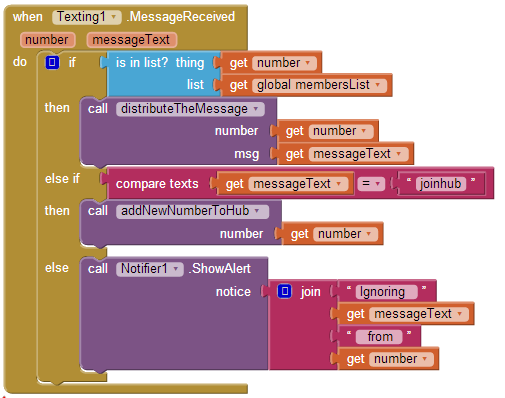
# 

# 

# Enhancements:

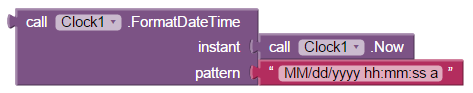
Here are some ideas for programming projects that could be done to enhance this app.

* **Abstraction:** Break the *if-else* code into more easily readable chunks by defining and using procedures with parameters. For example, if you name your procedures *addNewMemberToHub* and *distributeTheMessage* then your code would look like this:

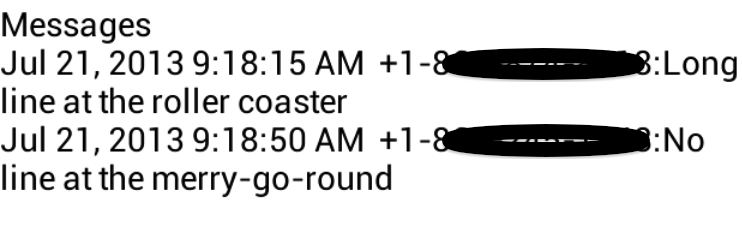


Notice that the ***call distributeTheMessage*** block has two slots for *arguments.* It needs to pass the *number* and *messageText* to the procedure. The procedure then distributes the message from that phone number. The ***call addNewMemberToHub*** block has one argument slot because that procedure needs to have the *number* so it can add it to the *membersList*.

* Modify the *distributeTheMessage* procedure so that it doesn’t distribute the message back to the sender of the message. HINT: You’ll need an *if-else* block for this.
* Add a *Clock* component to the app so that you can display a *timestamp* when a message is received. The Clock component contains methods that enable you to get the current time (*Clock.Now*) and format it:



Here’s what the timestamped messages would look like if you use these blocks:



* Add a *TextToSpeech* component so that the new members can be ‘announced’ to the user when a *‘joinhub’* message is received.
* **Persistence:** Add a *TinyDb* so that the members of the hub can persist from one use of the app to another.
* **Advanced:** Modify the app so that it saves the members’ names, as well as, phone numbers of the members of the hub. HINT: Use a list of lists that includes the member’s name and phone number *or* use a delimited string such as: “5558885555:Joe” where the : separates the name from the phone number. For this latter approach you should design functions to get the name or the phone number parts from the string.
* **Advanced:** Modify the app so that a member can be removed from the hub. HINT: You will need to design a *protocol* for leaving the hub (e.g. ‘removemember’)
* **Longer-term (Advanced) Project:** Come up with your own variations of this app. For example, one variation might be to extend the app to have multiple hubs -- family, friends, etc. And, allow members to tag their messages with certain prefixes to indicate which distribution list should receive the message -- e.g. “family: The picnic is at 1 PM’.

***Nice work! Complete the Self-Check Exercises and Portfolio Reflection Questions as directed by your instructor.***