|  |  |
| --- | --- |
| This version of Map Tour will extend the original app in two ways: (1) by incorporating a ***LocationSensor,*** App Inventor’s GPS sensor, and (2) by incorporating ***TinyDB,*** App Inventor’s database.    The LocationSensor will allow us to extend the list of destination by adding one’s current location (as determined by the GPS sensor) to the list of destinations.    The TinyDB will allow us to persist our list of destinations between different uses of the app -- that is, the destinations list will be saved and restored when one quits and restarts the app.  **Objectives:** In this lesson you will   * create an app that uses App Inventor’s GPS sensor to determine the user’s location and use it as a destination; * learn basic concepts about databases data persistence; * incorporate a TinyDB database component into an app to permanently save app data on the device. | ScreenshotPart2.png  ***[Click to watch Preview Video](http://www.youtube.com/watch?v=Jq3IrFv58n0)*** |

## 

# Introduction: Persistent Data and TinyDb

Up until now, the data in our apps has been stored either in ***global variables*** or as the value of the ***properties*** of the app’s various components. For example, when you store a piece of text in a Label, that data is stored in the computer’s main memory, in its RAM or random access memory. And as we’ve learned, RAM is ***volatile***, meaning that any data stored there will be destroyed when the app is exited.

By contrast, data stored in the computer’s long-term storage -- e.g., on the phone’s flash drive -- will ***persist*** as long as the app is kept on the device. There are various ways to store data permanently on a computer. For example, you could store it in a file, such as a document or image file. Another way to store persistent data is in a ***database.***  App Inventor provides us a very simple, easy-to-use database in its ***TinyDB*** component. Any data that we store in the TinyDB, will not disappear when the app is exited. Instead, it will persist between uses of the app -- even if you turn off the device.

Before we learn how to incorporate TinyDB into our app, let’s spend a moment to learn about this very important component.

[](https://www.youtube.com/watch?v=VCv59tjUqd0)

(Click the image to start the [video](https://www.youtube.com/watch?v=VCv59tjUqd0).)

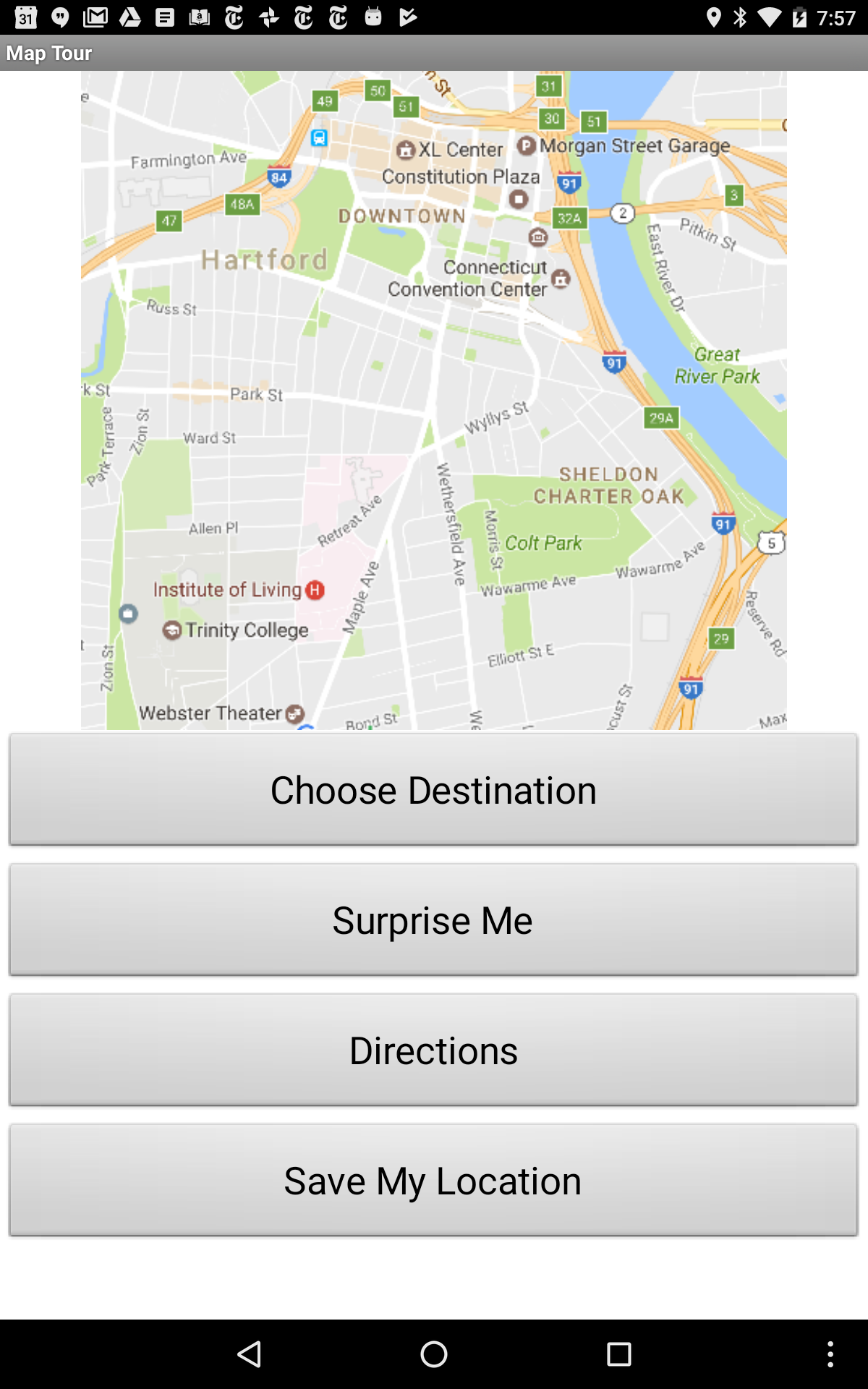
## Getting Ready

To get started, you can use the app that you created in the previous lesson or, if you prefer, you can [open App Inventor with the Map Tour GPS Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit3/templates/MapTour2017/MapTourGPSTemplate.asc) in a separate tab. When the project opens, use the Save As option to rename it ***MapTourGPS***. Then follow along with the following tutorial.

## 

## The User Interface

The UI for this version of the Map Tour app is very simple. It adds only a *Save My Location* button to the visible UI. For the non-visible interface, it includes the following components:

* TInyDB - for permanently storing addresses.
* LocationSensor - for tracking the device’s location.
* Notifier - for displaying messages.

**Add the *Save My Location Button***

Drag and drop a Button component onto the app, rename it *ButtonSaveLocation* and set its *Text* property to “Save My Location” and its *Width* to “Fill parent.” **Important: Set the *Enabled* property to *False* (unchecked).** We will enable this button only after the device has obtained a fix on its location from the Location Sensor.

**Add the *TinyDB Component***

Add a TinyDB component to the app. No need to rename it, since there is only one. It has no properties to set. We will code its behavior in the blocks editor.

**Add the *Notifier Component***

Add a Notifier component to the app. No need to rename it, since there is only one. It has no properties to set. We will code its behavior in the blocks editor.

**Add the *LocationSensor Component***

Add a *LocationSensor*  component from the *Sensors* drawer to the app. Set its *Enabled* property to *True* (checked) and set is *TimeInterval*  to 10000 milliseconds (10 seconds).

**NOTE:** If your device does not support GPS, you won’t be able to use the LocationSensor to acquire new addresses to add to the destinations list. In that case, here is an alternative way to add new locations to the destinations list:

* Add a *TextBox* to the app’s UI, named *TextBoxNewDestination*
* Code the *ButtonSaveLocation.Click* handler to use the value in the textbox to add to the destinations list.

By taking these steps, you will be able to use the TinyDB and the other features of the app.

# 

# Coding the App’s Behavior

Before working on the app itself, let’s review some information about App Inventor’s *LocationSensor.*

### GPS and the Location Sensor

App Inventor’s [Location Sensor](http://ai2.appinventor.mit.edu/reference/components/sensors.html#LocationSensor) (in the Sensors drawer) is a non-visible component that provides location data about the device’s ***longitude***, ***latitude***, ***altitude*** and ***street address***. It can also perform [geocoding](https://en.wikipedia.org/wiki/Geocoding), which is the process of transforming an address into its geographical coordinates (latitude and longitude). You can read more about the Location Sensor in App Inventor’s glossary.

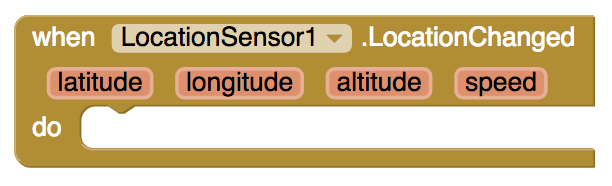
A mobile device can detect its location in one of three ways:

* Using its built-in [GPS](http://en.wikipedia.org/wiki/Global_Positioning_System) sensor, which acquires a fix on its location from GPS satellites. This is the most accurate but, ideally, requires that the phone have a clear shot of the sky so that it can receive readings from at least 3 GPS satellites. This is accurate within a few meters but uses the most battery power.
* Using a Wifi signal from surrounding Wifi router. The phone’s location would be the latitude and longitude of the router. This might work indoors and uses less battery power.
* Using the Cell ID -- i.e., signals from surrounding cell towers. This is least accurate but uses the least power.

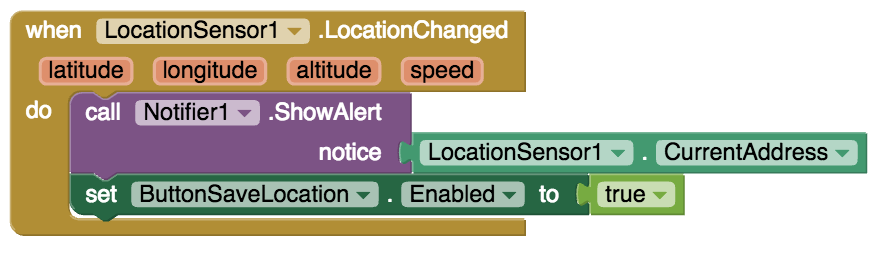
This app will use the Location Sensor to determine the device’s street address.

## Adding Your Location to the List

If your device has a GPS capability, it is easy in App Inventor to add your current location to the list picker’s list. The *Location Sensor*  has the following event handler, which will fire whenever the device’s location changes:



When it fires it provides the device’s latitude, longitude, altitude and speed. When it does fire, we will simply display the device’s current street address in a notification and we will enable *ButtonSaveLocation.* **NOTE:** Handling the *ButtonSaveLocation* in this way will guarantee that the user does not save empty (“”) addresses to its destination list. The enabling of the button will indicate that the device has obtained a fix on its location. This may take a while.



## How This Works

If you have set the Location Sensor’s *TimeInterval* to 10000 and the *DistanceInterval* to 0 (the default value), it will check for location updates every 10 seconds. Setting the distance interval to 0 means that the device doesn’t need to move in order for its location to be updated. The larger the time interval, the less frequently will the sensor check for location changes. The larger the time interval, the less battery power is consumed. The larger the distance interval, the less often the *LocationChanged* event will fire.

When the *LocationChanged* block fires, this means that the device has gotten a fix on its location and will display the current street address. It is at this point that the user should be able to add the current address to the destinations list. **NOTE:** This may take a while depending on where the device is -- it will work best out of doors.

**NOTE:** The Location Sensor can be a tricky component to use and its performance is very dependent on the type of device you have and where you are -- being indoors in a classroom is not optimal. So to get this part of the app working you may want to package the app and take the device outdoors. Also, make sure that the device’s GPS sensor is on. You can also try changing the default setting for the *TimeInterval.* A shorter interval will force the sensor to check for a change in location more often. Here is information about [Location Sensor](http://ai2.appinventor.mit.edu/reference/components/sensors.html#LocationSensor).

**If the LocationSensor does not work for your device or location, see the enhancement at the end to add a Textbox for adding a location instead.**

## Coding *ButtonSaveLocation*

When the user clicks the *ButtonSaveLocation* we want to add the device’s location to the destinations list, using the ***add items to list*** block:



This block will add a new item to the *destinations* list. We also want to update the *ListPicker’s* elements. This leads to the following coding of *ButtonSaveLocation.Click:*



## Running and Testing the App

Although this app will work in the *App Inventor Companion* it is preferable to package the app (the ***Build*** tab). That way you can test the app outdoors in case it is unable to get a fix on its location inside the classroom,

Here’s a test to run. Once the *ButtonSaveLocation* becomes enabled, the device should report its current address through a notification message.

1. Press ButtonSaveLocation to save the current location to the destinations list.
2. Press the *Choose Destination* list picker to confirm that the new address has been added to the list.
3. Quit the app (by hitting the Android ***Back key***).
4. Restart the app and look for the saved address on the *Choose Destination* list.

If you perform these steps, you should notice that the address you saved is no longer on the destinations list. Why is that? The reason is the the *destinations* list is a global variable and, as such, it is store temporarily in the app’s RAM (random access memory). When the app is quit its RAM memory is (effectively) erased. So when the app is restarted, the destinations list is re-initialized to its original state.

**If the LocationSensor does not work for your device or location, see the enhancement at the end to add a Textbox for adding a location instead.**

## Persisting the Saved Locations with TinyDB

As you saw from that test run, the locations that you save while running the app do not persist between uses of the app. To remedy this, we will store the destinations list in a TinyDB. There are two TinyDB actions that need to be taken in order to save and restore the locations:

1. When the app starts up, it needs to read the destinations list from the TinyDB.
2. When the user clicks *ButtonSaveLocation* the app needs to store the destinations list to the TinyDB.

### Inputting the Destinations List from TinyDB

The algorithm for inputting the destinations list is complicated by the fact that the first time the app is used, there won’t be a destinations list in the TinyDB.

**destinations ← INPUT\_FROM\_DB(“DESTINATIONS”)**

**if destinations = empty\_list**

**{**

**destinations ← [ "Connecticut Capitol", “Hartford Atheneum", “Trinity College”]**

**STORE\_TO\_DB( “DESTINATIONS”, destinations)**

**}**

**ListPicker.Elements ← destinations**

The algorithm begins by reading the destinations list, stored under the tag “DESTINATIONS”, from the TInyDB. If this is the first time the app is being used, then the attempt to read the “DESTINATIONS” tag from the TinyDB will return an empty list -- the app hasn’t yet stored any data under that tag. In that case we need to construct a new *destinations* list and store it in the TinyDB under the “DESTINATIONS” tag. We then set the ListPicker’s *elements.*

In App Inventor, we want to place this algorithm in the *Screen1.Initialize* block, which will fire as soon as the app starts up.

As you see here, *TinyDB1.GetValue* block is used to retrieve data from TinyDB using the “DESTINATIONS” tag and the *TinyDB1.StoreValue* block is used to store data into TinyDB using the same tag.

### Inputting the Destinations List from TinyDB

The algorithm for storing data into TinyDB will go into the *ButtonSaveLocation.Click* handler, which now looks like this:

## Running and Testing the App

Package (build) and run the app again and use it to save some new destinations to the List Picker. Now, when you quit the app (or turn off the device) the destinations you saved will be re-loaded from TinyDB into the app when it starts up again. By storing data in a database, TinyDB, the app is able to ***persist*** data that would otherwise disappear when the app stops running. The saved data will remain associatedwith the app as long as the app is on your device.

# Enhancements and Extensions

1. **Add a Location with a Textbox:** If your LocationSensor does not work on your tablet for your location, you should add a location using a TextBox instead. Add a TextBox and a Save button to your UI. The Save button code should be very similar to your SaveLocation button but it should add TextBox.Text to the destinations list instead of the LocationSensor address. Make sure you save the changed destinations list in the database and to the ListPicker elements.
2. **Monitor Your Device’s GPS.** As you may have experienced in using this app, the performance of your device’s GPS may vary widely depending on your location and your environmental conditions. If your device’s GPS is turned on, then obtaining a fix on its location (from satellites) will work best when you have a clear view of the sky. It will also depend on how long it takes the device to get a locate at least 3 GPS satellites, which depends on a variety of factors, including whether the device has been powered off for a long time or not. To monitor this write a procedure that will display some of the data that you can obtain from the Location Sensor. Write a procedure named *displayGPSdata* that gets called whenever the *LocationSensor.LocationChanged* event fires. The procedure should display (in a label or notifier) the value of certain properties, a list of properties is [available here](http://ai2.appinventor.mit.edu/reference/components/sensors.html#LocationSensor). Here’s a good set of properties to display:

**Accuracy** in meters

**Providers** - shows available providers where ***gps*** means satellites, ***network*** means cell towers and/or Wifi access points, ***passive*** means a fix may be obtained from other apps on the device.

**Provider Name** - which type of provider is being used to gain location fix.

**Latitude, Longitude.**

Displaying these and other data from the Location Sensor will give you an inside look at how it works.

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