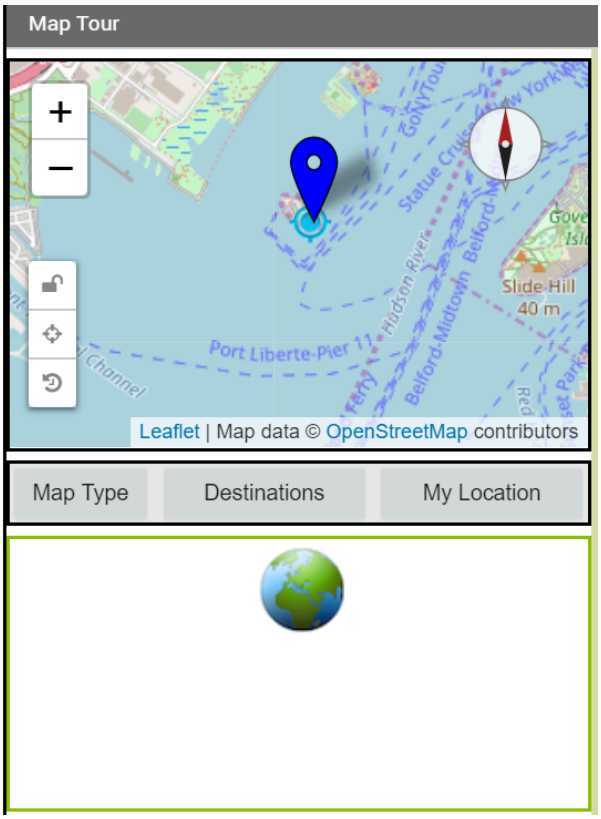
Overview: Map Tour with TinyDB Tutorial

This version of Map Tour will extend the original app in two ways: (1) by allowing users to add new destinations to the map tour, and (2) by adding ***TinyDB,*** App Inventor's database to store the destinations permanently. TinyDB will allow us to make our list of destinations persistent -- that is, the destinations list will be saved and restored when the user of the app quits and restarts the app.

Objectives: In this lesson you will learn

- how to add items to lists;
- how to use a Notifier for input;
- basic concepts about databases and data persistence;
- how to use a TinyDB database component to permanently save app data on the device.

Short Handout



Getting Started

Open App Inventor and open your Map Tour Project from the last lesson.

Designing the User Interface (UI)

The only changes to the User Interface of this app are to add a Label at the top that informs the user that they can now add locations to the tour, a Notifier component, and a Storage/TinyDB component.

UI Component	Name	Properties
Label	Label1	Text - Press and hold on map to add a destination Width - Fill Parent

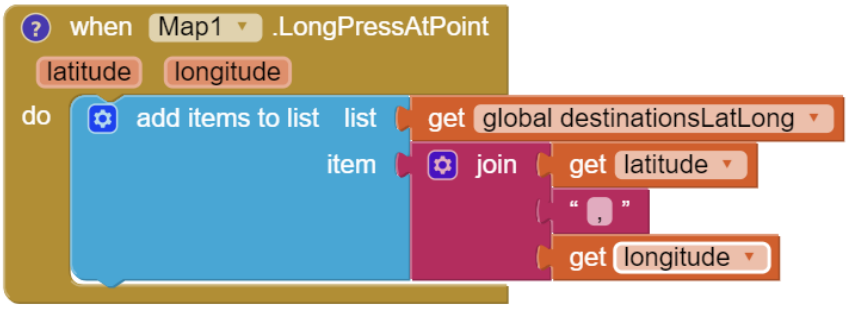
Notifier	Notifier1	No changes
Storage/TinyDB	TinyDB1	No changes

Coding the App

This app will add new destinations to the map tour when the user does a long press at a point on the map. This will trigger a notifier dialog box where the user can type in the destination name. The latitude, longitude, and the destination name will be saved in the lists and then in a database.

Add New Locations in Code

We can allow users to add more locations to our map tour and the destinations lists by pressing down on a location on the map. Use the Map's LongPressAtPoint event to add the new item to the list destinationsLatLong using the Add Item to List block. Since this destinationsLatLong list contains text strings of the form "latitude, longitude", we need to create that text string using a text join block.

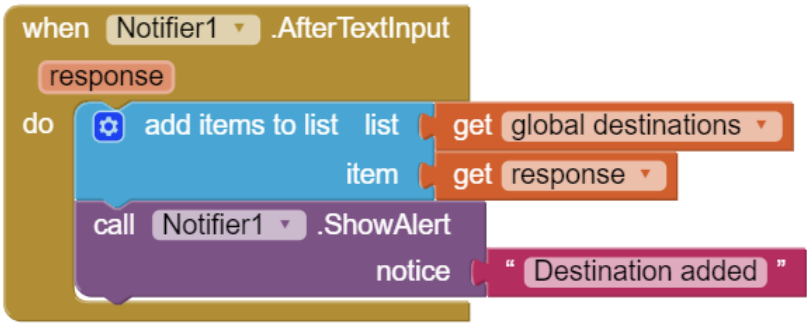


But this only gets the latitude and longitude. It does not add the destination name.

We can add a Notifier component to ask the user the Name of the destination. Drag in a Notifier component into the UI if you don't have one already. The Notifier's ShowTextDialog will pop up a box that can ask for a location name:



After the user types in a location name, it will be sent to the **When Notifier.AfterTextInput** event where you can add the user's typed in response to the destinations List. You can also pop up an alert to let the user know the destination was added.



A summary of the code is in the table below.

Event Handlers	Algorithms
Map1.LongPressAtPoint	-Add get latitude joined with a "," to get longitude, to your list destinationsLatLong. -Call Notifer1.ShowTextDialog to ask for the destination name.
Notifier1.AfterTextInput	-Add the user's typed-in response to your destinations list. -Call Notifier1.ShowAlert with the notice "Destination added"

Optionally, you could also add a marker at a Long Press by using the Map1.CreateMarker block. This block requires you to set up a new variable for the new marker and set its value to this block.

Running and Testing the App to see Temporary Storage

Although this app will work in the *App Inventor Companion,* package the app using the ***Build*** tab and install it on your device so that we can test what happens when we restart the app (or you could restart it in Companion). Note that you will not be able to Build on iOS devices, so you will not be able to test the database function on iOS, but everything else will work through the iOS companion.

Here's a test to run on any device.

1. Long Press to create a new marker and save it to the destinations list.
2. Press the *Destination* list picker to confirm that the new destination has been added to the list.
3. Quit the app (by hitting the *Back key*).
4. Restart the app (or reconnect with Companion) and look for the saved destination on the *Destination* list.

If you perform these steps, you should notice that the destination 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 stored temporarily in the app's RAM (random access memory). When the app is closed, its RAM memory is (effectively) erased. So when the app is restarted, the destinations list is re-initialized to its original state.

However, we can save the lists in a Storage/TinyDB database at the end of Notifier.AfterTextInput and restore them in Screen.Initialize to have persistent data.

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 closed. Instead, it will persist between uses of the app -- even if you turn off the device.

Tiny DB Blocks

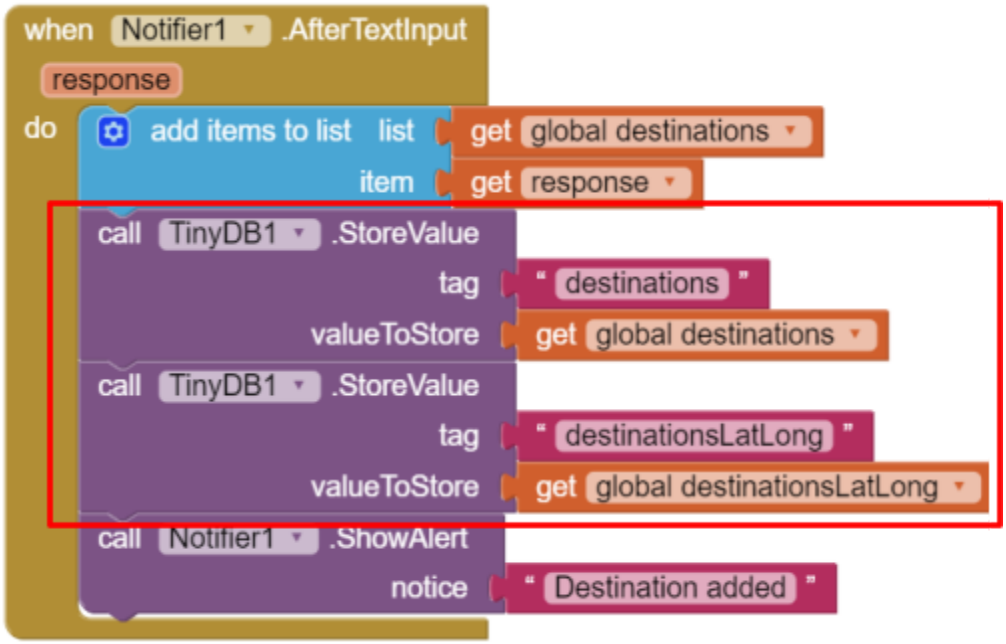
TinyDB has a **GetValue** and a **StoreValue** block to store and read data on the user's device. Data is stored under a tag which is just a Text block where you can type in any name to describe the data. The tag you use to store data must match exactly (including whether it is lowercase or uppercase) with the tag you use to get that data back out of the database. There is also a useful **ClearAll** block if you want to reset the database and erase everything in it.



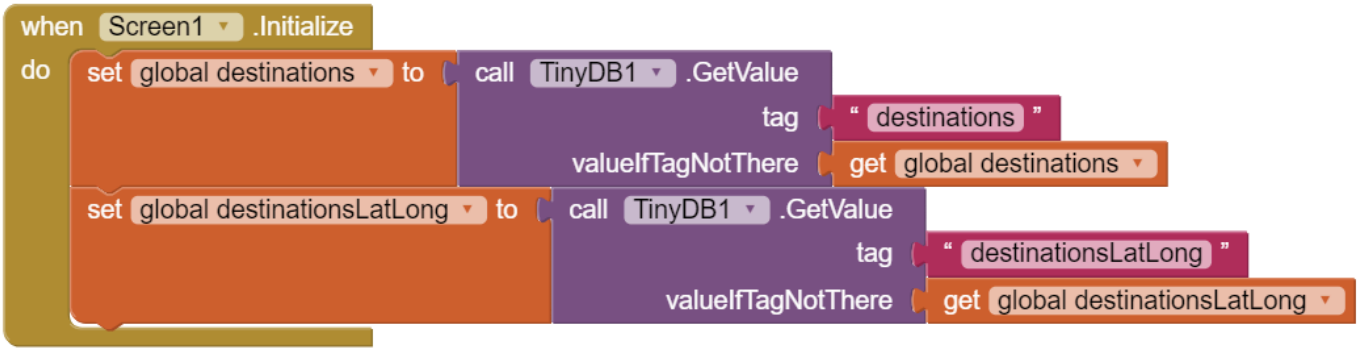
Persisting the Saved Locations with TinyDB

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.

When the user long presses the map, and the new destination is added to the lists in Notifier.AfterTextInput, the app needs to store the changed destinations and destinationsLatLong lists in TinyDB. The TinyDB stores and identifies data by using something called a **tag**. By using the same **tag** name for .StoreValue and .GetValue, you can save and retrieve your data. In this case, you are storing the entire destinations and destinationsLatLong lists with two separate tags in two different areas of the device storage! You must make sure the tag names match in GetValue used in Screen.Initialize and StoreValue here so you get the exact value you saved.



When the app starts up, it needs to read the destinations and destinationsLatLong lists from the TinyDB. This can be done using a TinyDB.GetValue block. If the destinations are not in the TinyDB then, they can be read from the current lists.



A summary of these changes are below:

Event Handlers	Algorithms
Notifier1.AfterTextInput	-At the end of the code in this procedure, store the two lists in TinyDB.
Screen1.Initialize	-Set your destinations and destinationsLatLong lists to values stored in TinyDB; put a get block for the original lists in the if not found area. Make sure you use the same tag names as in the Store blocks.

Testing the App

On Androids devices, 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 associated with the app as long as the app is on your device.

On iOS devices, you cannot build, but you can test the other functions of this app using the iOS Companion app.

Test your app with the following inputs. See if the outputs match the expected outputs.

Inputs	Expected Outputs	Actual Outputs

Long press on Map	Notifier asks for destination name and then creates marker.	?
Click on Destinations	The new destination is on the list.	?
On Android devices, close app, open app, and then click on Destinations.	The new destination is on the list.	?

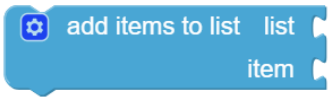
Enhancements

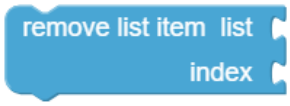
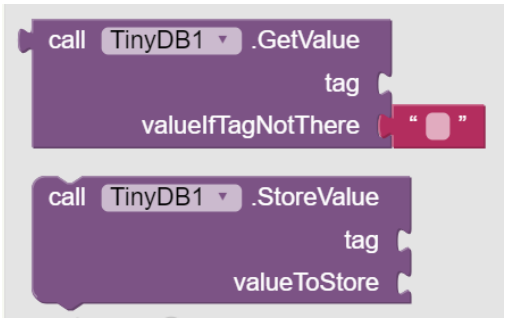
Your instructor may ask you to do some or all of the following enhancements for your Map Tour app.

1. **Delete Locations.** As you are testing your app, you may have added a lot of locations on your map tour that you do not want. You could delete the data stored for the installed app in your device under Settings/Applications Settings or by calling TinyDB.clearAll in your code, but in this enhancement you will add a Delete ListPicker button that lets you choose a location to remove from your lists and update the database. Here are the steps you need to do:
    - Add a ListPicker to the UI to Delete destinations.
    - In ListPicker.BeforePicking, set the ListPicker.Elements to the destinations list.
    - In ListPicker.AfterPicking, use the **remove list item** block from the Lists drawer to remove the item at the ListPicker.SelectedIndex from both of the lists (destinations and destinationsLatLong). Save both lists in TinyDB. Use Notifier.Alert to tell the user the destination was deleted.
    - Refactor your code to add a saveToDB procedure to save both lists in TinyDB and call it from ListPicker.AfterPicking and Notifier.AfterTextInput.
2. **My Location:** If you have a device and location where GPS works. when you click on the My Location block, add that location to the destinationsLatLong lists using the Add Item to List block and use the Notifier.ShowTextDialog to get the location name for the destinations list (this will call the already written Notifier.AfterTextInput procedure).

Summary

Here are some new App Inventor List blocks that you used in this lesson and their AP pseudocode versions for the AP exam.

App Inventor Blocks	AP Pseudocode
	APPEND (list, value)

	REMOVE(list,i)
	No database code in AP exam.

Vocabulary Review

Review the following new vocabulary and concepts in this lesson. If you don't know what these mean, look back through the lesson.

- Add item to list
- Remove item from list
- Database
- TinyDB
- Persistent data
- Any Component

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