Table of Contents

Giving Your App Long-Term Memory: TinyDB and TinyWebDB
Storing Persistent Data in TinyDB
Retrieving Data From TinyDB
Sharing Data with TinyWebDB
Storing and Retrieving Data with TinyWebDB
Storing Multiple Tags
Setting up a Web Database
Summary
Giving Your App Long-Term Memory: TinyDB and TinyWebDB
A database is an app’s long-term memory. The data in the database is called **persistent data** because even when you close the app and re-open it, the data is still available--it persists.

As an example, consider the *No Text While Driving* app that auto-responds to texts the phone receives by sending back a “sorry, I’m busy” message. Such an app should let the user customize the response message. When the user enters a new custom message, the expectation is that the new message will be saved Persistently--in a database. If the user changes the custom message to, "I’m sleeping, stop bugging me," then closes the app, the message should still be "I’m sleeping, stop bugging me," when the app is reopened.

Persistent data is different than the variables and properties an app uses for its computations. The data stored in variables and properties is transient, or short-term memory, and when you close an app the data stored there is lost. For the custom response sample, you might store the custom message entered by the user in the Text property of a TextBox. If you don’t transfer that data to persistent storage, and the app is closed, the custom message entered by the user will not reappear when the app is re-opened. This would make the users of your app quite unhappy. For an app to remember data in its long-term, persistent memory, it must explicitly store the data there--it must copy the data from a property or variable into the database. The general progression is:

1. The user enters some data into a TextBox component
2. The user clicks a "Submit" button
3. The app copies the data from the TextBox to the database.
4. When the app re-opens, the app copies the data from the database back into the TextBox.

**Storing Persistent Data in TinyDB**

App Inventor provides two components to facilitate database activity: TinyDB and TinyWebDB. TinyDB is used to store data persistently on the phone, while TinyWebDB allows you to store data in a web database that can be shared amongst phones. TinyDB is simpler to use, so we’ll explore that first.

You store data--transfer it to long-term memory--with the TinyDB.StoreValue block:
A tag-value scheme is used for database storage. The tag identifies the data within the database, while the value is the data itself. The tag is just a key that you’ll use later when you want to retrieve the data from the database. In the sample, the data is tagged with the text “responseMessage”. The value (valueToStore) is the custom message stored in the Text property of the ResponseLabel component (it is what the user has entered).

The StoreValue transfers the data from the short-term memory to the database. After the StoreValue is executed, the database will have the following entry:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>responseMessage</td>
<td>I’m sleeping, stop bugging me</td>
</tr>
</tbody>
</table>

An app might store many tag-value pairs, and each value can be either a single piece of information (a text or number), or a list of information. we’ll see an example of a list value later.

The data is stored in the database for one reason: so it can be re-loaded when the user closes and later re-opens the app.

**Retrieving Data From TinyDB**

You retrieve data from the database with the TinyDB.GetValue block. Often, you'll call GetValue in the event-handler Screen1.Initialize. Screen1.Initialize is a special event-handler that is triggered when the app launches-- the blocks you put there will be executed when the app opens. Here are the blocks for loading the database data when the app opens:
TinyDB1.GetValue is called with the tag, "responseMessage", the same tag that was used when the data was stored. The database finds the value associated with the tag and sends it back.

It is a common pattern to first place the value returned from GetValue in a variable, because you need to check if the database actually has data for the tag you request. If it is the first time the app has been launched, or if the user has never entered data (e.g., a custom response), then the database won’t have the requested data.

When the database has no value for a given tag, GetValue returns an empty text object. You can check for an empty textbox by checking the length of the value returned. If the length is 0, the database has returned the empty text. For this sample, it is important that the data from the database only be put in the ResponseLabel if it is non-empty-- otherwise you’d overwrite the default response message.

**Sharing Data with TinyWebDB**

The TinyDB component stores data in a database directly on the Android device. This is appropriate for apps that don’t need to share data, like the SMS texting auto-response app. That app might be installed on many phones, but each instance of the app uses its own isolated database as one phone doesn't need to access the custom response of another.

Many apps do need to share data amongst phones-- Facebook and other social networking apps are prime examples. The MakeQuiz/TakeQuiz app (Chapter N) is another-- in that sample, a person on one phone creates a quiz and stores it in a web database so a person on another phone can take it. For such data-sharing apps, the database must live on the web, not the phone.

The TinyWebDB component is used for apps that need to share data. TinyWebDB stores data on a web database so more than one phone can access it, as illustrated by this diagram depicting the MakeQuiz/TakeQuiz scenario:
But where on the web is the data stored? By default, the TinyWebDB component stores data on a web (database) service setup by the App Inventor team and accessible at [http://appinytynwebdb.appspot.com](http://appinytynwebdb.appspot.com). This default database is for development only-- any App Inventor app can store data there and you have no assurance that another app won’t overwrite your data!

Fortunately, it is relatively easy to create your own TinyWebDB database service. No programming experience is necessary-- you can just download a sample, tweak a name in a configuration file, then upload it to the “cloud” with Google’s App Engine. We’ll show you how to do this later in this chapter (Section 20.5).

Once you setup your database, the next step is to add a TinyWebDB component to your app. The TinyWebDB component has just one property-- the ServiceURL. By default it’s set to the default web database. But if you create your own web database service, you can set the ServiceURL property to its URL:

```
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyWebDB</td>
</tr>
<tr>
<td>ServiceURL</td>
</tr>
<tr>
<td><a href="http://usfwebservice.appspot.com">http://usfwebservice.appspot.com</a></td>
</tr>
</tbody>
</table>
```

Here, the ServiceURL is set to http://usfwebservice.appspot.com, which is a web database that Professor Wolber setup for his student’s apps. Once the ServiceURL is set, all TinyWebDB.StoreValue and GetValue calls will be sent to the specified URL.

**Storing and Retrieving Data with TinyWebDB**

Whether you use the default TinyWebDB web database, or one you create, the method of storing and retrieving data with the TinyWebDB is very similar to that of TinyDB. The StoreValue operation is exactly the same. For instance, with the quiz creator app you store the list of questions in the following manner:
Retrieving data from a web database is a bit different with TinyWebDB compared to TinyDB. With TinyDB, the GetValue operation is immediate because your app is communicating with a database (file) directly on the phone. With TinyWebDB, the app is requesting data over the network.

For this reason, TinyWebDB uses an asynchronous scheme. Your app will first request the data it wants by calling GetValue. GetValue just makes the request, without expectation of immediate satisfaction. Your app, in fact, might go on to perform other function blocks and even handle other events before the request is fulfilled. To illustrate, check out the difference between the TinyDB.GetValue block and the TinyWebDB.GetValue block:

The TinyDB.GetValue block returns a value immediately, and thus a plug appears on the left-side so that value can be placed into a variable or property.

The TinyWebDB.GetValue doesn’t return a value immediately, so there is no plug on the left-side. Instead, when the web service fullfills the request and the data arrives back at the phone, a TinyWebDB.GotValue event is triggered. Your job as a programmer is to code that event-handler to process the data. An event-handler like GotValue is sometimes called a call-back procedure.

For instance, suppose you had stored a list of questions for a quiz and you wanted to load the questions into your app when the app launches. In Screen.Initialize you’d just request the data:

You’d then code the GotValue event-handler to process that data when it arrives to the phone:
The requested data appears in the argument valueFromWebDB. In this sample, the event-handler checks to see if it is indeed a list— if no questions have been stored it will be an empty text string, which should not be placed in the variable QuestionList. If the returned value is indeed a list, the data (the questions) is placed into the variable QuestionList.

**Storing Multiple Tags**

Because TinyWebDB works in an asynchronous fashion, you must handle the GotValue event with care. Specifically, if you request more than one type of data, e.g., both the questions and answers of a quiz, both requests will be answered and not necessarily in the order they are requested.

Fortunately, the GotValue event provides an argument, TagFromWebDB, which identifies the request. If you requested both the questions and answers of a quiz in the Screen.Initialize:

```
when Screen1.Initialize
  do
    call TinyWebDB1.GetValue
      tag text questions
    call TinyWebDB1.GetValue
      tag text answers
```

the blocks for GotValue might look like this:
Note that GetValue is called twice, once for the tag "questions", once for the tag "answers". Even though the questions are requested first, you cannot be certain the web service will send the questions first. Instead, in GotValue, you must check the tagFromWebDB. If it is indeed “questions”, the valueFromWebDB is placed in QuestionList. Otherwise it is the answers, and the valueFromWebDBis placed in AnswerList.

**Setting up a Web Database**

The easiest way to setup an App Inventor web database is to copy the files of our open source sample, change one name in a config file, then upload the code to Google’s App Engine. No programming or complicated server/database setup is required.

1. **Download App Engine.** The first step is to download and install Google’s App Engine Launcher. This desktop tool significantly eases the task of testing web apps and services, and deploying them to the App Engine server.

2. **Download Web Database Sample Code.** Next, download the zip file at http://appinventorapi.com/program-an-api-python/ and unzip it. You should see three files: app.yaml, main.py, and index.html, and a directly ‘images’.

3. **Register with App Engine.** App Engine is a cloud computing tool so it has a web component as well as the desktop “Launcher” tool you downloaded. You can reach the web part of App Engine at http://.appspot.com. Follow the instructions to register. Note that you’ll need a cell phone number to register--Google uses this to safeguard against fraud (after all, they are providing each user with gigabytes of data as well as an enormous amount of processing power).

4. **Create a new Project in App Engine.** Create a new project. You’ll need to provide a name that is unique to the App Engine system.

5. **Modify the Config File.** Back on your desktop, open the app.yaml file that
came with the zipped sample. The first line looks like this:

application: appinventordb

Change the text ‘appinventordb’ to the name of the project you created at App Engine.

6. Open App Engine and Deploy. Open the App Engine Launcher desktop tool. From the menu, choose File | Add Existing Application, and choose the folder where your modified sample code resides. Then click the deploy button. Enter the user name and password of your App Engine account, and your web database service will be deployed to the cloud.

7. Test your service with a browser. Though the purpose of the web database service is to handle requests from App Inventor apps, the app also provides an “administrative” interface for humans. If you named your App Engine app “mywebdb”, open a browser and visit mywebdb.appspot.com. If all is well, you’ll see a simple interface for storing and retrieving data.

8. Test your service with an app. You’re ready to write an app that uses your new web database. Open an existing app or create a new one, then make some calls to TinyWebDB.StoreValue and TinyWebDB.GetValue. Remember to also provide a TinyWebDB.GotValue to actually process the data returned.

9. Administer the service and database. App Engine provides a number of tools at appspot.com for maintaining your web database. Each app you create has a dashboard which will show you things like the number of “hits” your app (web service) receives. You can also examine the data stored in the database and delete items when necessary.

Summary

App Inventor makes it easy to store data persistently through its TinyDB and TinyWebDB components. Data is always stored as a tag-value pair, with the tag identifying the data for later retrieval. TinyDB is used when it is appropriate to store data directly on the device. When data needs to be shared amongst phones (e.g., a multi-player game), TinyWebDB must be used. TinyWebDB is more complicated as you need to setup a call-back procedure (the GotValue event-handler), and a web database service as well.