|  |  |
| --- | --- |
| In earlier apps that we designed in this course, we used TinyDB to store and retrieve information on our physical device (phone or tablet). But in this lesson, we will build a simple Clicker App that will store and retrieve information on the Web.    Imagine a teacher asking the class a question and students each voting on it. We want to design an app that can not only store the results centrally but also allow the teacher and the students to view the results in real time. Objectives: In this lesson you will learn to:  * create an app that can be used to poll individuals and store responses on the web; * understand the concept of ***centralizing and sharing Web data;*** * grasp the difference between ***synchronous*** and ***asynchronous*** operations; * use a TinyWebDB database; | Screenshot (69).png  ***[Click to watch video](https://youtu.be/WiJsuuEYeIE)*** |

# Introduction: The TinyWebDB Database

One of the big ideas in this course is “Data and Information.” Specifically, in this unit we want you to understand the trade-offs when representing information as digital data and how the processing of data can be used to help users gain insight and knowledge. To those ends, we will now introduce a simple, web-based database called TinyWebDB.

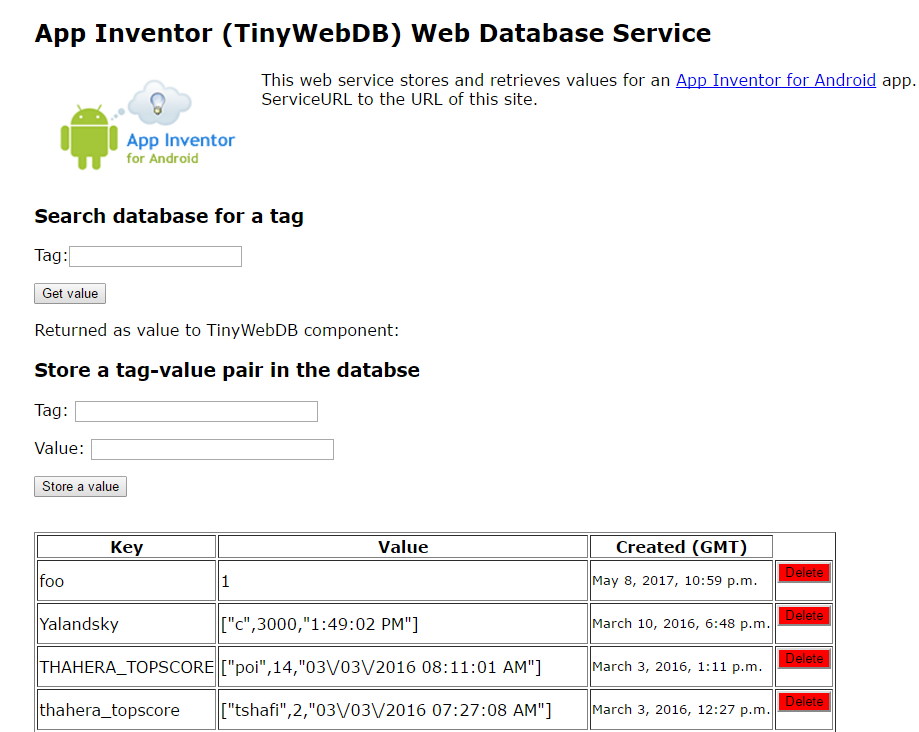
TinyWebDB is a non-visible component that can be used to store and retrieve data values in a database located on the Web. It can be found in the Palette’s *Storage* category. Whereas TinyDB stores data only on the device running the app, TinyWebDB can be shared among multiple users and multiple devices running the same app because it is online in the cloud.

TinyWebDB is well integrated into App Inventor. The TinyWebDB URL property identifies the URL of the database where information can be stored and retrieved. App Inventor has a default TinyWebDB that can be used.

In today’s Internet connected world users of mobile devices can benefit by accessing data stored external to the device. In this Clicker App we will poll a set of users and store the results of the poll in a centralized TinyWebDB database. We will then display the results of that poll in real-time on our mobile devices.

TinyWebDB is a minimalist, free, public web-based database that is well integrated into App Inventor 2. Although there is a default instance of this database attached to App Inventor

(http://cpsc110-db.appspot.com), your instructor may ask you to use a different instance with your classmates.



Consider the chunks of data shown in the image, below. Note that each DB record consists of three fields: a key (or tag), a value, and a timestamp of when the datafield was originally created. The red **Delete** buttons shown on the right appear only in the human-web interface at <http://appinvtinywebdb.appspot.com/> to the DB and are not part of the DB, itself.

## The DB stores two types of records, individual data items or *heterogeneous* lists. Consider the “foo” key (or tag) shown in the DB below. The numeric value 1 is associated with this tag. This means that if you *query* the database with a tag of “foo,” the DB will return the value 1 as the answer. In comparison, the “Yalandsky” tag has a list of data associated with its tag. This list contains the string “s”, the number 3000, and a timestamp of 1:49:02 PM. We say the list stored under a tag can be heterogeneous because the list items can be of different data types.

Notice the keys are case sensitive in a TinyWebDB as demonstrated by the last two records shown in this database.

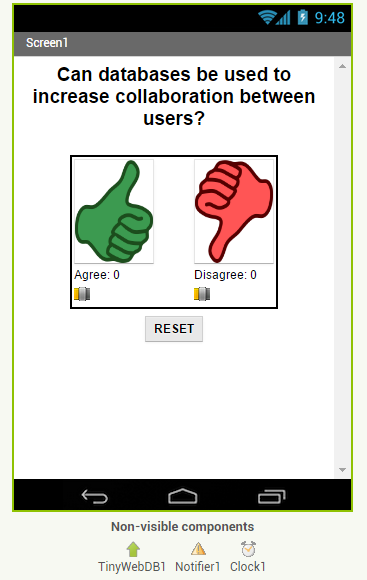
Since the database is public, anyone can read or write to it. That means that a malicious user can delete or alter your data. Furthermore, other users can accidentally change your data if they inadvertently use the same tag. **For this reason, you will have to be careful to use *unique tags* when developing your app.**

# Getting Ready

Start App Inventor with the [Clicker App Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit6/templates/ClickerApp/ClickerWebDBtemplate.asc). Once the project opens use *Save As* to rename your project *ClickerTinyWebDB.* You should see a User Interface similar to the one shown above.

## The UI

Most of the UI is already built for you. Add a User Interface/Notifier, Sensors/Clock, and Storage/TinyWebDB (not TinyDB!). Set the TimeInterval property of the Clock to 3000.

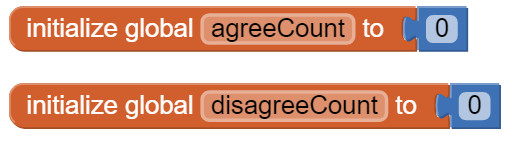


Note the three non-visible components shown above, the TinyWebDB, Notifier and Clock/Timer will be used in the App.

# Code Walkthrough

We will now walk through and discuss the details of the important coding blocks in this app.

## Define Variables



# Programming the Buttons

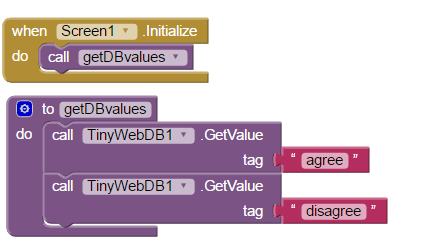
Reacting to the button events in this app is fairly simple and straightforward. When the user presses the “Thumbs Up” or “Thumbs Down” buttons, we want to add 1 to our data and store the new data back in the database. The RESET button is mostly used for debugging purposes. It will only go into the teacher version of the app which we will build as an enhancement in a future lesson. Likewise, the Clock component will only be used in an enhancement.

## buttons.png

The storeDBvalues procedure shown in the above blocks is something you have to code. The Reset button in this version of the app is primarily there to help you in your debugging process. However, in a later enhancement, we will build a “Teacher” version of this app. For that enhancement, the Reset button will become a functional and important feature for the Teacher user.

## Coding the Screen1.Initialize Event

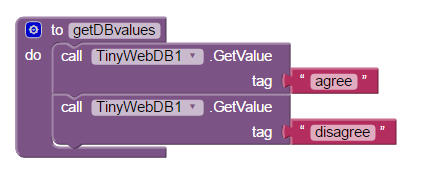
When the screen first initializes, we want to fetch data immediately from the database to initialize the screen with whatever values are currently stored. Even though the tags shown in this example use the simple strings “agree” and “disagree”, you will have to modify these tags (perhaps by adding your school’s name to them) to make sure that the tags you are writing to the TinyWebDB are all unique. You must do this for every tag in this lesson.



# Retrieving Stored Data Asynchronously from TinyWebDB

We often store data in a database so that it can be retrieved later. In this app, polling data are stored inside a TinyWebDB database. The app periodically loads this information and displays it to the user.

## TinyWebDB Get Value: Asynchronous Retrieval



Like the TinyDB, data stored in TinyWebDB is associated with a unique ***tag***. To retrieve a piece of data, you must specify its tag, as shown in this block. In this app, we use the tags ***“agree”*** and “**disagree**” as the tags for storing the number of responses to a poll, each in numerical form.

**NOTE:** Remember that even though in our example, above, we have shown these tags to be simple, single English words, to avoid conflict with your classmates, you may need to prepend a unique prefix to these (and any other) tags to avoid a database conflict with your neighbors.

## Synchronous vs. Asynchronous Retrieval

Notice the difference between the *TinyWebDB.GetValue*block and the *TinyDB.GetValue*

**

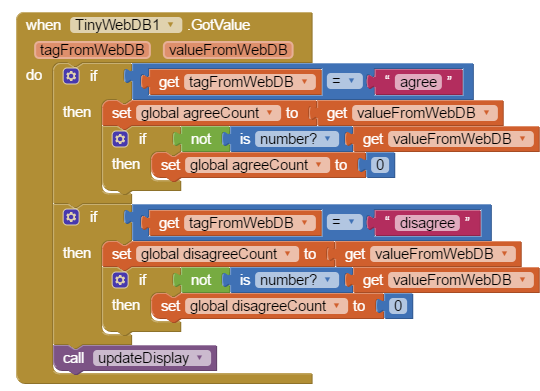
Expression

The *TinyWebDB.GetValue* is a ***statement*** *block* whereas the *TinyDB.GetValue* is an ***expression* *block.*** This difference is crucial to understand. In the case of TinyDB when you retrieve a value you would use the expression block to immediately assign it to a variable, as we did in the previous lesson:

*TinyDB.GetValue* is an example of ***synchronous retrieval***. This means the retrieval happens immediately. *Synchronous* means *at the same time.* So the retrieval happens at the same time as when the *GetValue* block executes. The TinyDB is stored on the device’s permanent storage (i.e., flash drive) and the retrieval is nearly instantaneous. While the retrieval of data from TinyDB is many milliseconds slower than retrieving data from the device’s RAM memory (which would take microseconds), the difference would not be noticeable to the user.

However, synchronous retrieval is impossible over the Web. In order to retrieve something from the Web your app must send a request over the Internet. This takes time, usually a few milliseconds but that’s not the same as *at the same time.* Therefore, retrieval over the Web is ***asynchronous retrieval*** -- i.e., *not at the same time.*

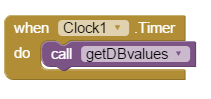
Moreover, things could go wrong when retrieving data over the Internet. For example, your Wifi connection could be slow or dropped. Or, the Web site storing the data could be down. Or, any number of other ***I/O problems*** could result***.***  Because of this, the app cannot wait for the data to be received. Instead, when the device receives the requested data it will trigger an event and it will notify your app. Your app must use an ***event handler*** to retrieve the data. This is the ***TinyWebDB.GotValue*** *event handler block:*

****

This block will be triggered as soon as the device receives the data from the Web. It figures out which query was answered -- i.e., which tag has been requested -- and updates the internal data member associated with the corresponding response. If we were expecting a number but no number was returned (as may happen the first time the app is run or in the case of an error), we set the data element to a safe value of zero with the use of the “if not is number ...” statements shown.

# Use a Polling Algorithm to Automate the *Update* Process

When the user presses the “Like” or “Dislike” button, how can we be sure that the data we currently have stored in the app is up-to-date? Is it possible that the values stored in the database have changed since we last updated? We could require the user to press a button before addition more polling information to get the latest data from the database. But is there a better way? We could try to automate the process by which data is updated from the database to the individual devices.

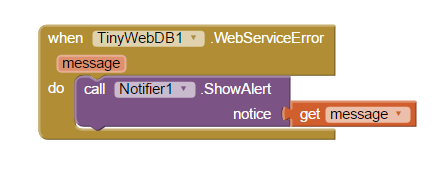


To do this, we can use a Timer to call the procedure call “getDBvalues” method periodically. Every time the timer fires, we could poll/ask the database to refresh our data. We need to program the timer to fire frequently enough so that that data remains well synched across all devices. But if we choose a time interval too short, the various devices will constantly be polling the database, eventually slowing it (and our app) down. By setting the timer to 3000 ms (3 seconds), we can balance these two concerns.

With this polling method using a periodic timer, we can do a fairly good job of keeping all our devices in synch. But is it possible to do even better? What if we could program the database to tell us when its data changed? This is known as a “PUSH” feature. Unfortunately, TinyWebDB does not provide a PUSH feature. But in an advanced lesson we will explore this concept further with a more modern database.

## Error Handling

If TinyWebDB returns an error message of some sort (perhaps an Internet issue, a server problem, etc.), we want to display the error message that is received by the app to the user. We, therefore, encode the *WebServiceError*  handler as shown using a *Notifier*.



# Testing the App

This app is best tested by forming a group of students where everyone in the group uses the same tag names for the shared database. Make sure that when one of your peers votes, the latest data shows up on everyone’s screen after 3 seconds. If you vote at the same time, your vote may get overwritten. This is called a <b>race condition</b> which we will fix in the next lesson. With the 3 second delay, you may also find that the data is not in sync for everyone in the group. In the first enhancement below, you will try different TimerIntervals to reduce these problems.

# Enhancements

**Timer Interval:** Try different TimerIntervals for the Clock in the UI. A lower number like 250 means that the app data will be updated every 250 milliseconds. A higher number such as 3000 means that the data will be updated every 3 seconds which might result in race conditions and the data not being in sync across devices.

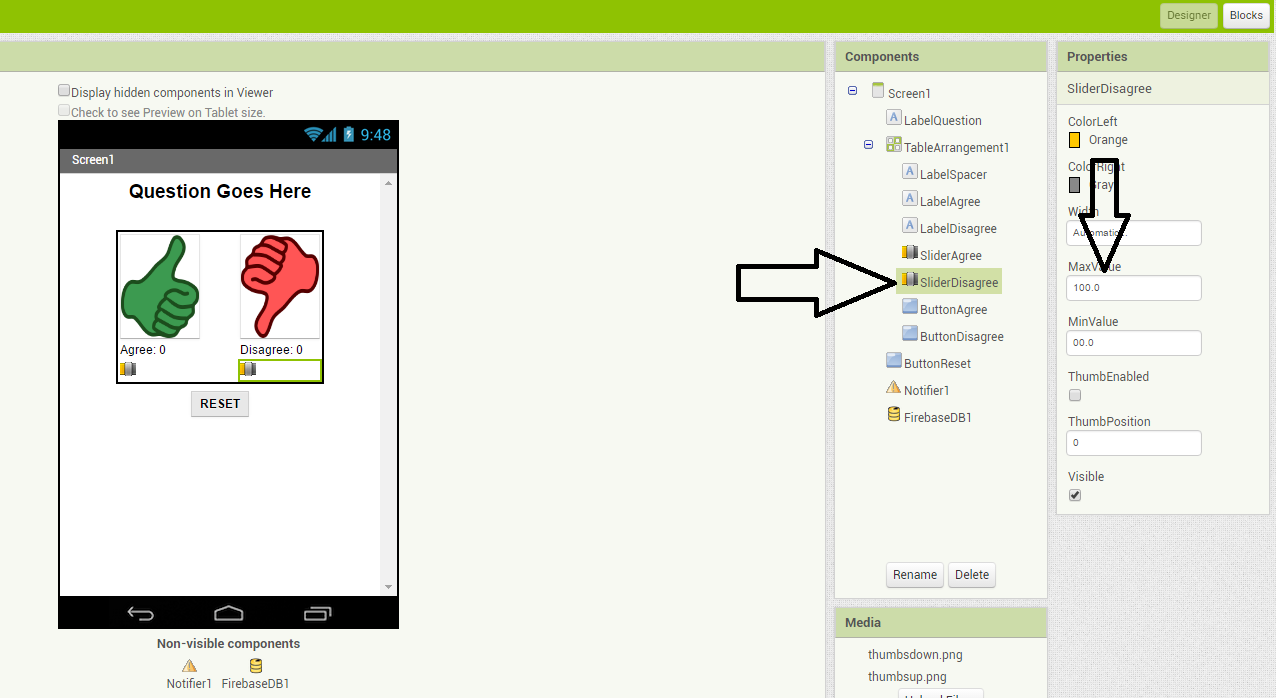
# Enhancement #1: Create a Histogram Using the Thumb Switches

This will likely be the first app for which you will use the *Slider (Thumb Switch)* component. The component is a bit complicated to use so you are encouraged to read [this documentation](http://ai2.appinventor.mit.edu/reference/components/userinterface.html#Slider) or watch this [video](https://www.youtube.com/watch?v=cm2-kVcWTuw&feature=youtu.be) before proceeding.

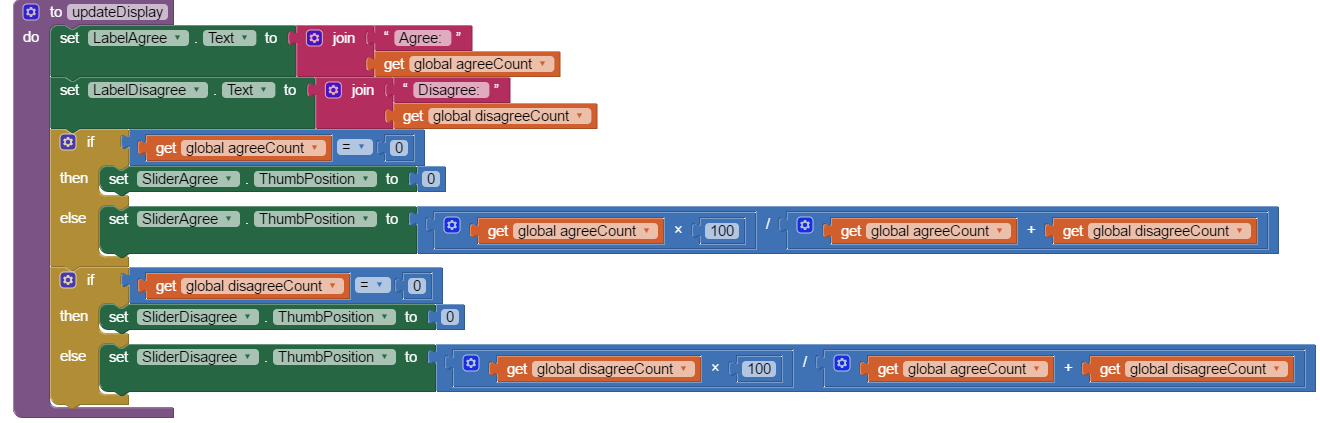
The sliders or thumb switches are most frequently used to allow the user to set the value of some property by moving their thumb on a sliding scale. For our Clicker app, we will be using this component in reverse - to create a ***Histogram*** (or ***bar chart***) based on the ratio of “Agree” and “Disagree” votes recorded by the app.

**Programming the Sliders**

When we add the two Slider components to the User Interface, we have to be sure to set the MaxValue property to 100 since our display will be based on a percentage:



The general idea for setting the Slider ThumbPosition property is to divide the number of “Agree” or “Disagree” votes by the total number of votes and then multiplying by 100 to determine the percentage of “Agree” and “Disagree” votes.



We have to be careful not to divide by zero (which will cause a run-time error) when the app is first started, when the number of “Agree” and “Disagree” votes are both zero. Using IF statements can allow us to set the value of the slider position to zero when the total votes is zero. We have not shown these “IF” blocks; you have to figure them out for yourself.

# Enhancement #2: Allow Users to Vote Only Once

Modify the app so that the Clicker only allows the user to vote once (hint: there is an Enabled property for buttons). The updating feature should still continue to work with the timer. Add re-enabling the voting buttons when the user hits reset.

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