Itty Bitty City Code Companion – Arduino IDE

WELCOME!

What the Itty Bitty City Code Companion brings to you!

The Itty Bitty City Code Companion gives even more opportunity to enjoy Itty Bitty City at a more in-depth level. In this Code Companion, you will see how you can change the way projects function, or even make a new project. With Itty Bitty City you can make intelligent, technology-based projects, and with the Code Companion, you can explore even more.

What we are going to do with Itty Bitty City Code Companion

In the Itty Bitty City user manual, you can follow the basic steps to program your projects from our ready to use Arduino IDE project files and editor. With Code Companion we are going to look at the code that comes with Itty Bitty City to understand how it is a set of instructions that program your projects, how it can be changed and how you can change the way your projects work.

What does 'Code' do?

Part of what you do when you assemble your Itty Bitty City electronics projects using a mBattery, mCookie modules, and sensors, is to make a micro-computer. Like any computer, it needs instructions or code on what to do. Without this code, the micro-computer or CPU found in the red Core Module has no idea what to do or how to operate.

What does code look like?

Code looks like a list of words or sentences each on its own line. Each line of word or sentences is called a line of code. Lines of code often include numbers and punctuation.

Where do we write these lines of code?

Lines of code are written in a program on a computer. This program is called an "IDE editor" which has special functions such as checking the code, setting the communications mode to the project's computer and setting which type of computer board you are using.

The IDE Editors from Microduino also come with added libraries of pre-made code for the Microduino modules and sensors including those that come with Itty Bitty City. **This makes it easy** to add new code to your program.

The 'program'.

All the lines of code together that form a single project are called its 'program'. You may have heard people referring to program that they need to make or modify for a computer. A program are the lines of code and can be opened in your editor, which we already discussed', or it can be the program already stored in the Microduino computer (CPU) in your project.

A word of good advice…. as you make or modify a program, you can save the program as a file on your PC. However, it is best to always make a new name for each file if you save it to your PC so you don't lose the original file.

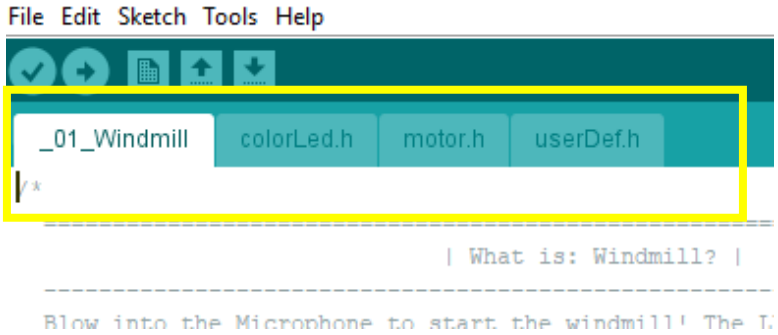
Let's do something…

OK, so we covered some of the basics. Now let's get some of this going to see better what you've been reading about. We are going to start by exploring the code for the Itty Bitty City Windmill project. So please get it ready by opening the Arduino IDE program, and let's see how to change the way it works!

By now, you have already confirmed the proper operation of the *standard* Windmill project. We can now continue to look at some of the details of the Windmill's code and make some changes!

If you like, you can also follow along on the pages of the Itty Bitty City Manual, pages 19-26.

You will need to have your IDE editor running and the Windmill file open, and it looks something like this:



Notice the four tabs highlighted in the yellow square above. The white highlighted tab is simply the current tab we are looking at. The other tabs will in turn also become highlighted when selected. Only one tab however can be selected at a time. A program can have many tabs. It is up to the *programmer* (that is YOU now) as to how many tabs there are and the names of the tabs. The tabs are simply a way to flip from one section of the program to another. We will come back to these tabs shortly.

Notice below that the first section of text is a light grey. That is because all text that have '/*' in the beginning and end with this '*/' are notes and automatically are in grey text. Notes are very handy, if not essential, to inform the reader or programmer about on this program runs. It is like a little notepad right in the program.

In the Windmill program the first section says how the Windmill runs and what should expect when it does run.

Read the text below the sections that have headers reading:

[What is: Windmill?]

[You will]

Read through these two sections to see what the Windmill does and how to make it function normally.

The next section is headed with the words:

```
| Change the code yourself! |
```

Now this is where we start to really see how this program is organized. The text in this section tells us:

> Click the userDef.h tab to change below values to your own liking. :)
>
> VOICE_MIN: Minimum noise level to activate the windmill and LED.
> MOTOR_SPEED_MAX: How fast the windmill spins.
> TIME_RUN: How long the windmill and LED will be on.
> BRIGHT_MAX: Max LED brightness.

Notice the reference above to the **userDef.h** tab in the above text. The original programmer is telling us that you can change the values, in this case we call the variables, to change how the Windmill operates.

We said we would come back to the tabs and now is that time. Click on the Windmill tab again, and under the 'Questions?' heading, you see now in green/black/blue text the first lines of 'code' that read like this:

```
#include <Microduino_Key.h>
#include"userDef.h"
#include"colorLed.h"
#include"motor.h"
```

The program above is establishing (sometimes called 'calling out') what is to be included on this and on any other specific tab pages to make the Windmill project function.

For now, we will bypass the meaning of the remaining text at the bottom of the Windmill tab page as we are just starting out.

Now let's select the tab called 'userDef.

Once this tab is selected, this is where you can make changes to the Windmill program!

In the top of the page below is where we tell the Core Module/CPU what we plugged into the hub, and where we plugged in to (page 21 of the Itty Bitty City Manual). *Don't worry about the first line below.* For example, notice the second line says the color LED is plugged into HUB on pin socket 12 (yellow square). The third line says the microphone is plugged into HUB socket A6 (green square). This is setting up the hardware just as it is shown in the Itty Bitty City manual page 21, (and another example of calling out.) You can change which socket the LED and Microphone plug into in the HUB *as long as you change the values shown below* to match.

```
#define DEBUG     0     //Serial monitor debugging ON/OFF.
#define PIN_LED   12    //ColorLED pin.
#define PIN_MIC   A6    //MIC Sensor pin.
#define VOICE_MAX 1023  //Maximum noise level to activate the windmill and LED.
#define LED_NUM   2
```

We won't make any changes for now until we understand the meaning of variables, and when a program is written it almost always includes sets of them. It makes programming much easier to write and to experiment with different values. *A variable is a specific value that you, as a programmer, sets. It can also be easily changed.*

For example, suppose you want to repeatedly refer to the mathematical term 'pi' in a program. Well, instead of writing 3.1415926535897 each time you just define pi once as 3.1415926535897 each time in your program. Sometimes there are hundreds of variables. To setup a Variable, in its simplest sense, you set a 'label' followed by a 'value'. The 'label' becomes the easy to remember name of variable and it is the label that is used throughout a program. So, our example with 'pi' it might look like this:

```
##define pi   3.1415926535897
```

It would be hard to have to change the same variable value throughout a program (remember it is not uncommon for a program to be millions of lines long!). So by using a label called 'pi' you can change the value of the variable in one location and the label will have the new value everywhere it is used in the program.

Back to our Windmill… at the lower portion of tab userDef.h there are four number of variables defined:

```
#define VOICE_MIN 400
        //Minimum noise level to activate the windmill and
        LED.  -Increasing this means you//will have to be
        louder to activate the windmill.

#define MOTOR_SPEED_MAX 100
        //How fast the windmill spins. Fastest speed is 255.
        A negative value will make
        //the Motor spin in the other direction. Note: 255 is
        very fast. Be careful!

#define TIME_RUN  10*1000
        //How long the windmill and LED will be on.
        Changing the number "10" to "20" means
        //it will spin for 20 seconds. Do not change the
        "1000".

#define BRIGHT_MAX  128
        //Max LED brightness. Max brightness is 255.
        Minimum brightness is 0.
```

In Arduino IDE coding the term '#define' means we are going to define a new variable. In this case, we see the label of 'MOTOR_SPEED_MAX is defined and has a variable value of numeric 100.

Notice also that on the lines of code for the variables, there are '//' followed by text which are in grey. The '//' means that the text following is not code. It is comments from the programmer to help other programmers who are using or revising the code, to know or remember the details of the variable. It is sometimes critical to let other programmers know what the code is doing and why as it is not always obvious. However, in the files of Itty Bitty City projects these comments are directed to us, the users, to act as helpful information.

For example…
On the line for: #define BRIGHT_MAX 128
The comments say the maximum variable value you can use is 255 and the minimum is 0. By the way if you set it to 0 , no light will come from the LED.

As you can see, you can have a fascinating time changing your variables. Each time you change a variable and want to see its effect on your windmill project, you need to upload the program to the Windmill using the same process as you did on pages 23-25 in your manual.

Some things to try…

- Make the windmill turn backwards
- Make the windmill go faster
- Make the Windmill stay on longer

Tab - Recap

Although we used the Itty Bitty City Windmill project as our example, all the Itty Bitty City projects use the same tab approach. The left most tab, whose tab name includes the

project name, gives an overview as to the project's purpose code. The right most tab, userDef.h, contains the variables that are easy to modify, upload and run in your Itty Bitty City projects.

There are the other tabs, between the project name tab and the userDef.h tab, that we have not talked about on the Windmill project; colorLED.h and motor.h. These tabs contain more sophisticated code that sets up the use color LED and windmill drive motor we just adjusted on the userDef.h tab. Look at the following;

In the tabs 'colorLed.h' and 'motor.h", the code contained there is not nearly as easy to understand as the variables. These pieces of code are more sophisticated requiring a higher-level understanding of code. The code in these tabs are essential for proper operation.

Changing variables, as you now have seen, is not as difficult as you may have thought. You may be very pleased to see that changing the way an ITTY BITTY City project operates is not so difficult, and just doing this can give you hours of fun and experimentation! For those who want even more of a challenge, continue on. Remember, with Itty Bitty City there are no bad mistakes, just learning experiences.

More Fun, -adding an additional motor.

So now you have seen some easy ways to change how and Itty Bitty City project runs. -Here's more. Back on the '_01_Windmill' tab, for example, in the grey text area below, it says you can add a second motor!

```
                    | Brainstorming Ideas |
-------------------------------------------------------------------------------
1. What else can you use to control the Windmill instead of the MIC Sensor?
   A button? Light Sensor? Remote control?
2. Can you add another Motor to the other side for TWO MOTORS?!
   That'd look pretty weird.
```

Next, if you check the motor code in the 'motor.h' tab we see that BOTH of the motor sockets on the Motor module are supported in the program code:

```
#endif
Motor MotorLeft(MOTOR_PIN0A, MOTOR_PIN0B);
Motor MotorRight(MOTOR_PIN1A, MOTOR_PIN1B);

unsigned long motor_timer;

void motorRun(int left_speed, int right_speed)
{
  MotorLeft.Driver(left_speed);
  MotorRight.Driver(right_speed);
```

Now just plug in your second Itty Bitty City motor into the second socket of the Motor module. It works!

Let's do it! (More Fun)

A frequent process in coding, at the learning and hobbyist level, is to use pieces of code that already exists. If you want to experiment with making your own project, which we

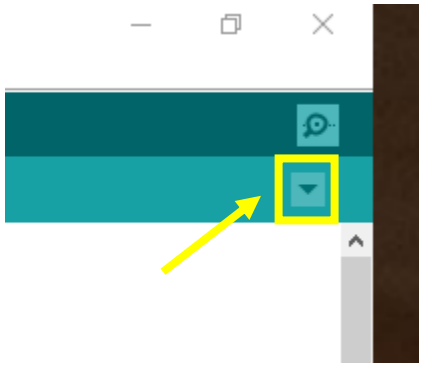
encourage you to do, you can copy/paste section code from known good programs into your own program.

For fun, let's add a buzzer to the Windmill that came with Itty Bitty City that is not used in the Windmill project. By adding a buzzer, we can play some music! Before proceeding, make sure you open both the IDE **Windmill** project and the **Piggy Bank** project files, sometimes called the sketch.

First go to 'File' on Windmill and click on Save As. Type in _01_Windmill_w_buzzer and click save. You will see an altered name change on upper left hand the tab.

Next, to add the music selections into the Windmill project from Piggy Bank, you will next need to add a 'music.h tab' in Windmill.

On Windmill, select the down arrow button in the upper right corner of the IDE editor screen.



Select 'New Tab' and name the tab below as 'music.h' in the lower right-hand corner. A new 'music. h' tab will appear above.

Next, copy the entire page from the Piggy Bank music.h files, and paste it into the Windmill's newly created music.h page.

With the above completed, we need to make changes to the Windmill sketch. Look at the Piggy Bank program and select the left tab entitled, '_05_piggy_bank'.

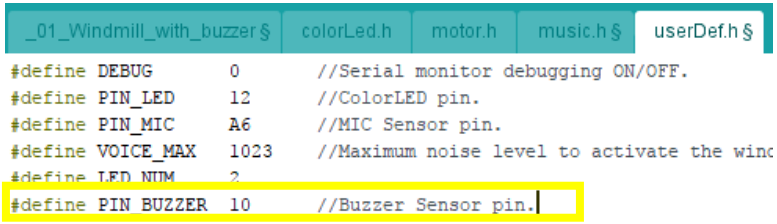
Scrolling down the sketch, you will see some grey text and then some definitions. What we want to add first to our Windmill sketch is '#include "music.h"'. Copy that line from Piggy Bank project and paste into the Windmill project found under the _01_ with_buzzer tab like below:

```
#include <Microduino_Key.h>
#include "userDef.h"
#include "colorLed.h"
#include "motor.h"
#include "music.h"
```

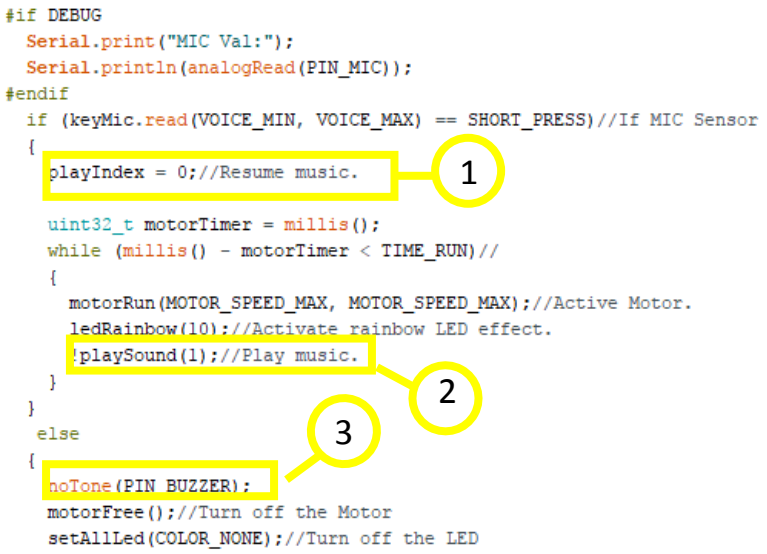
We have now 'borrowed' this line of code from the Piggy Bank project and pasted into Windmill.

Since we are adding a buzzer onto the Windmill, we need to next tell the Windmill computer/CPU where it will be connected to on the HUB. So, go back to the '_05_Piggy Bank' file now open in your IDE Editor, and go to the 'userDef.h tab' and find the line that designates which HUB socket the buzzer will be attached to.

We see the buzzer below is attached to socket 10 of the HUB. Copy this line and paste to the bottom of the list on the Windmill's list of settings in 'userDef.h' tab. It should then look like this:



The next steps are ones that may seem a bit hard if you are not yet familiar with actual coding. We will explain the changes and what they do. Each change is numbered and then explained as we go. Notice in the next illustration, we are going to copy/paste three additional inserts under the _01_Windmill_w_buzzer tab.



Let's assume you don't know code instructions or commands. Ok? Good! So, when we dig into this we are going to use our power of observation more than our actual knowledge of instructions/commands. So, let's make the following observations of the code we will be borrowing and pasting from the Piggy Bank project (first left tab), making sure you include the exact following commands **(in bold)** in the designated places as described above;

1. **playIndex = 0; //Resume music**.

   'playIndex = 0;' appears to be a command to make the music play (through the buzzer of course). Our hint is not the command itself but the 'note' after the '//' symbols that say resume music. We are going to place this line of code in a similar location in the Windmill code (thank goodness the code

structure of both projects is similar!). Note the “”;“ at the end of the command section. This means this is not the end of the string of commands for the computer to understand. Copy and then paste: ‘playIndex = 0; //Resume music.’ from Piggy Bank into Windmill as indicated.

2. **!playSound (1);// Play music**
   Based on the previous line of code we just looked at, can you start to sense what this line does? You look at the note (conveniently placed by the programmer) that says the command ‘!playSound’ means it is a command to play the music. Copy and then paste ‘!playSound (1);// Play music’ into the Windmill sketch as indicated. Note above the insertion of a semi colon between (1) and // highlighted in yellow.

3. **noTone (PIN_BUZZER);**
   Well, the music from buzzer can’t play forever so something must turn it off. Remember, devices (as in things like LEDs, motor, servos, buzzers, sensors), runs or don’t runs based on how the computer/CPU is programmed. Copy and then paste noTone(PIN_BUZZER); into the Windmill sketch as indicated.

   Finally, make sure you re-save your entire new Windmill scetch you have just created.

The Moment of Truth

Connect a buzzer to socket 10 of the HUB. Check that you are connected from your PC to the Windmill computer with the USB cable, AND that the mBattery is on (checking that the right most power indicator is red!). Upload the new IDE program to your Windmill. The Windmill should run as it did before, BUT now plays a song through the buzzer, with possibly a second motor. **Congratulations**!

P.S. If you want play a different song, change the ‘1’ in the “!playSound (1);//Play Music” command to any number from 2 to 11 (see the song names on the music.h tab)

Troubleshooting

If it did not work correctly, *do not panic*! Some trouble-shooting hints:

- Is the PC and Windmill connected?
- Is the battery in the Windmill have the red LED on?
- Have you selected the correct port?
- When you ‘upload’ does the editor say ‘done uploading’ in the lower status bar?
- If all is OK above, recheck your connections/wiring and try again.