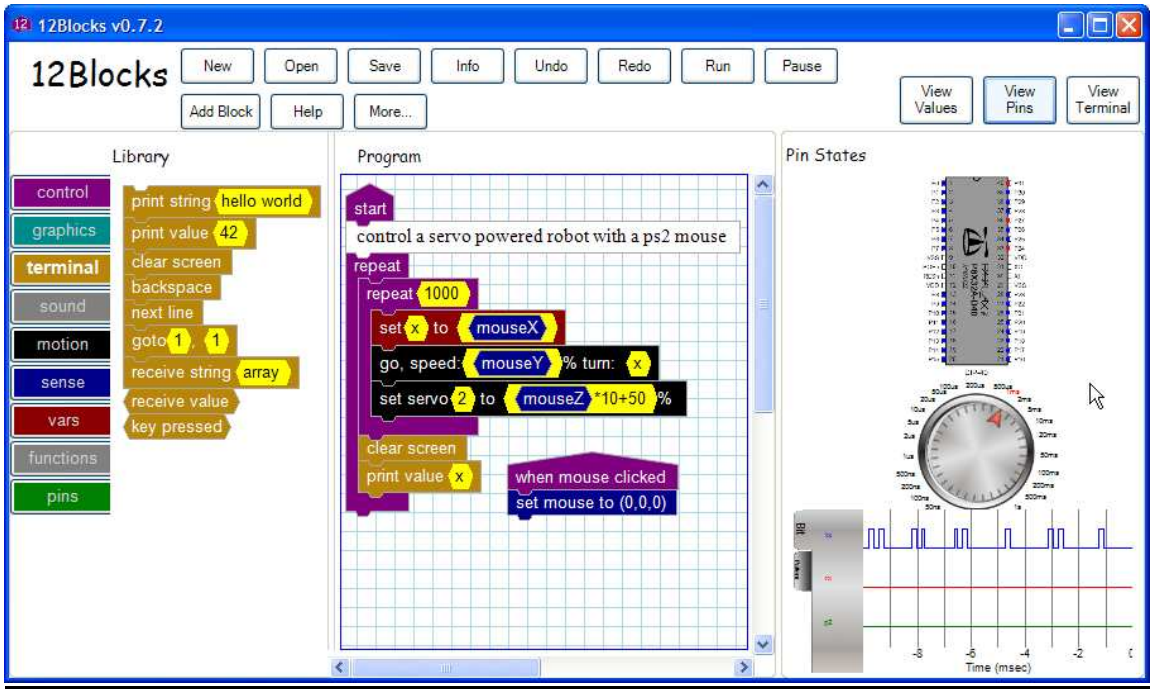
Programming the Propeller with 12Blocks

Hanno Sander V1.1



Introduction

12Blocks is a tool that let's you create programs by stacking blocks together. A library of over 100 powerful blocks let's you quickly build games, robots, and more. Just assemble blocks onto the worksheet and then debug your program with built-in graphical tools. 12Blocks is ideal for classrooms, students, hobbyists and professionals who want to quickly and easily build projects with the Parallax Propeller™.

With 12Blocks you can easily:

- write programs by assembling blocks onto a worksheet
- graphically debug programs with the built in visualizers
- use wizards to customize graphical sprites, vectors and more for your program
- change program parameters on the fly- without recompilation/reloading
- interface with common sensors, peripherals, actuators and devices
- easily integrate the Propeller with other PC applications

Program Library:
12Blocks makes it easy to share programs with others. Visit http://12blocks.com/howto and use the search tool to find a program that solves your problem. Users can upload circuit schematics, photos of their project, and the complete code for the program.

12Blocks is easy and fun enough to be used by a 5 year old, but extensible and powerful enough to be used professionally. In this lab you'll download and install 12Blocks in its 30 day evaluation mode and work through several sample programs that increase in complexity from a simple Hello World program to using arrays, functions, and event messages.

We'll cover how to :

- Graphically arrange blocks to create a program
- Edit block parameters- wizards exist for vectors, sprites, speech, wav files
- Change parameters while your program is running
- Comment your program with blocks that include links to files or websites
- Use multiple *start* blocks to write a multiprocessing program
- Write programs that output graphics, play music, control servos and more
- Define functions with arguments, local variables and return values
- Program with synchronous and asynchronous event handlers
- Customize the library by adding new blocks to implement new functionality
- Use Hardware definition files to define pin usage and customize the block library
- Share programs online with the 12Blocks community
- View and edit a file created by 12Blocks as SPIN code in the Propeller Tool
- Combine blocks and text as parameters for a block
- View and change variable values in a running program using an oscilloscope interface
- View pin activity using a graphical map and logic analyzer graph
- Interact with a program using the terminal
- Integrate with Python, Matlab, Excel, VB.NET, C# or other DDE/.NET aware applications

Videos:
If you prefer watching videos to reading this document, check out our video tutorials at: http://12blocks.com/videos

Prerequisites

1. Download 12Blocks from here: http://12blocks.com/download and install it.
2. Connect a Parallax Propeller to the PC using a USB cable.

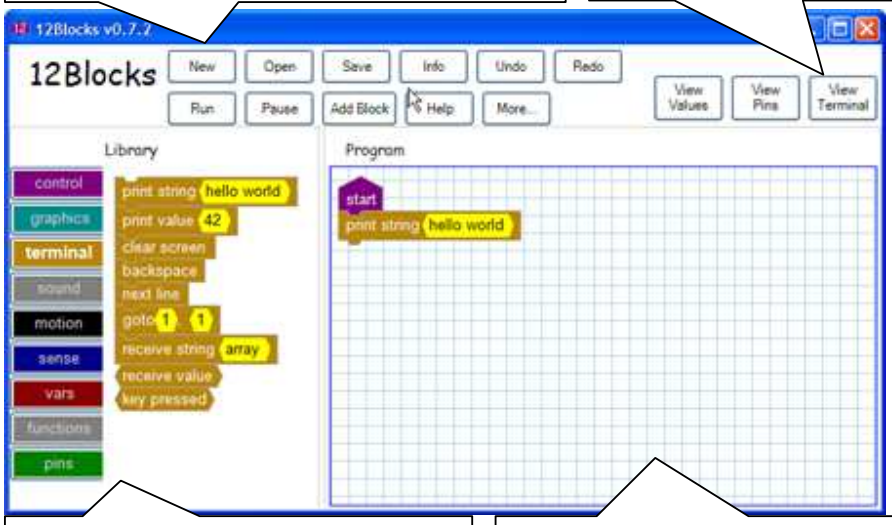
Hardware:
This tutorial was tested with a Parallax Propeller Demo Board available here:
http://www.parallax.com/Store/ProductSearchResults/tabid/127/ProductID/340/List/1/Default.aspx?SortField=ProductName,ProductName
If you're using something else, make sure it's using a 5Mhz crystal and is connected to your PC using a USB PropPlug. 12Blocks will run your programs on the first Propeller it finds- if you want to change this behavior, click the *More..* button and select *Ports* to configure which ports will be searched. If you require additional assistance check out our forums for help: http://forums.hannoware.com/

Program #1: Hello World

The first example shows how to print `Hello World` to the built in terminal. First get to know 12Block's interface- it consists of *Command Buttons*, *View*, a *Library* and a *Worksheet*.

Click on **Command Buttons** open and save files, run your program and more.

Views help you understand what your program is doing when it is running.

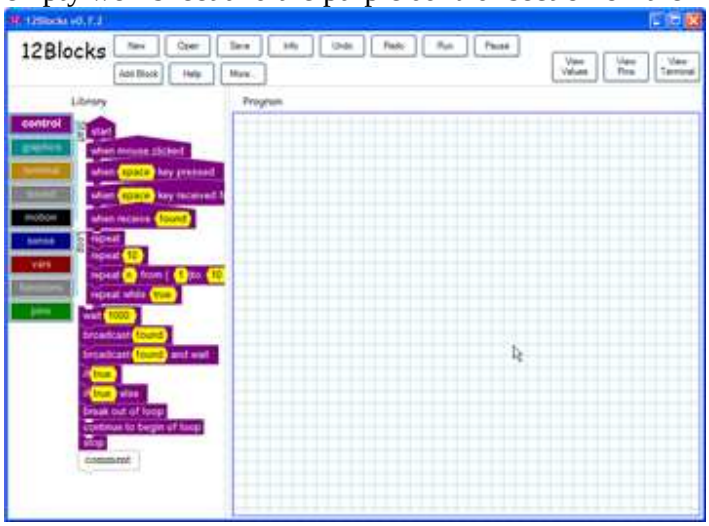


The **Library** of ~100 blocks is divided into sections. Each has blocks that can be dragged to the worksheet. Drag blocks back to library to return them.

The **Worksheet** is where you assemble blocks into programs.
Drag a block or stack of blocks around the screen. Move a single block from a stack by sliding it out to the left.

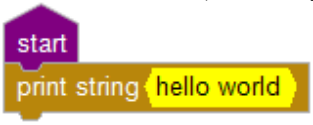
ta .

Start 12Blocks and choose "Start New Program" at the welcome screen. You should see an empty worksheet and the purple *control* section of the library.

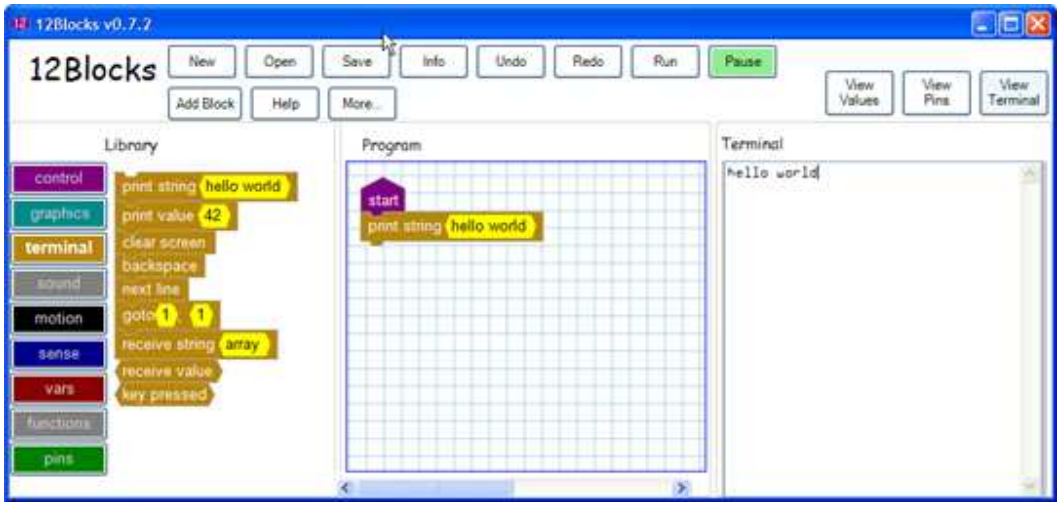


Your programs must start with one of the *start* blocks- they're drawn with a triangular top. So, start your program by dragging the *start* block to the worksheet. (Move your mouse to the *start* block in the library, hold down the left mouse button, move the mouse back to the worksheet and release the mouse button).
Your program in the worksheet should look like this:

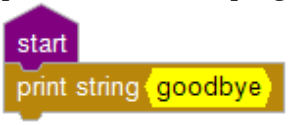
Now, you need to add a block which will print text to the terminal. Click the *terminal* button to show the terminal part of the library. Drag the "print string" block so it joins to the *start* block in the worksheet. (Drag it to the bottom of the *start* block and let go of the mouse button when you see the blue hint). Your program should look like this:



That's it, you've assembled your first program!
Now, confirm you've connected a working Propeller to the PC you're using and run your program by pressing the *Run* command button. 12Blocks will translate your program into a Propeller Spin file, compile that into Propeller instructions, load those to the Propeller, and establish a high-speed connection with the Propeller- all in less than a second! To view the output of your program click on the *View Terminal* button to open the terminal. You should see this:

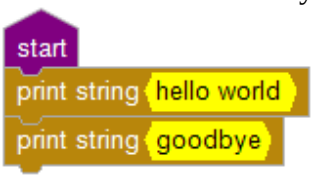


Now, make some changes to your program. Start by changing what text will be printed. Click on the yellow area- this is the *parameter* section of the block. Type `goodbye` into this box and press enter. Your program should now look like this:

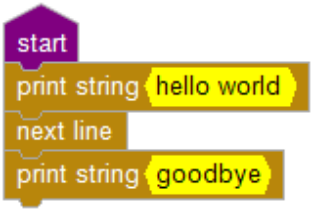


The terminal should now display `goodbye`. To print `hello world` before `goodbye` drag another *print string* block BETWEEN the *start* and the existing *print string* blocks. (Drag the

print string block from the *terminal* section of the library to the bottom of the *start* block, release the mouse button when you see a blue hint.) Your program should look like this:



Run your program- we're almost there! We just need to insert a new line between `hello world` and `goodbye-` like this:



When you're finished with your program, press the *Save* button. Name your file `Hello World` and hit the save button. To open your file in the future, press the *Open* button, select the `Hello World` file, and press the *Open* button.

Additional things to try:

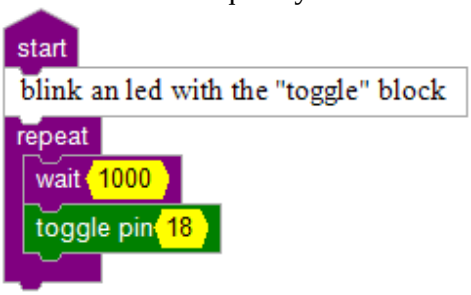
- If you make a mistake, click the *Undo* button to go back one action at a time. Click the *Redo* button to go forwards.
- Click the *New* command button to add a "New" worksheet.
- Close a worksheet with the "x" next to it's name.
- Click the *Help* command button to read the help manual.
- If you don't want to use a block anymore, drag it back to the library.

Directories:
At installation, 12Blocks will create a 12Blocks subdirectory in your `My Documents` folder. You should save files there to ensure the required include files are found. If you wish to save files to a different directory, make sure to copy files from the `MyDocuments/12Blocks` folder to the folder you intend to use.

Program #2: Use multiple processors to blink lights

Sample Programs:
Instead of assembling this program, you can load it from your tutorial directory. Click *Open* and make sure you're looking in the *Tutorials* folder. This program is called `lights.12b`

In this program you'll use three of the Propeller's 8 cogs (Parallax's name for processor) to blink lights with different techniques. The simplest way is to use the toggle block with a wait block which sets the frequency- like this:

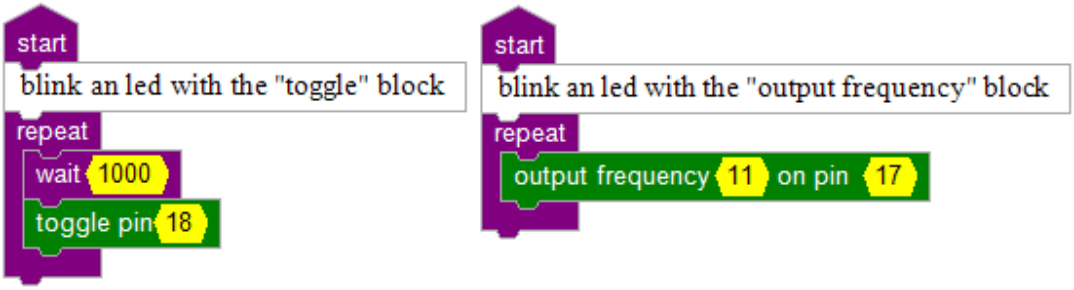


This program includes some new blocks:

- The white block is a `comment` block that we can use to document our programs. Just click on its text to edit it.
- The repeat block has two attach points, one at the bottom and one indented in it's inner part. The *repeat* block is one of several *loop* blocks that will run the inner blocks a set number of times before continuing. The simple *repeat* block will loop forever.
- The *wait* block will wait a specified amount of time- given in milliseconds, so this block will wait for 1 second.
- The *toggle* block is from the "pins" section of the library. Blocks in that section directly manipulate one or more of the 32 pins of the Propeller. You can attach things like lights, switches, sensors and motors to those pins. In our case, we're toggling pin #18, which on the Demo Board is connected to an LED- a thing that lights up when it's turned on. So, the *toggle* block will turn the LED *on* if it was *off*, or *off* when it was *on*.

So, this program will continually wait 1 second and then toggle the LED- making it blink.
Press the *Run* command button to see for yourself!

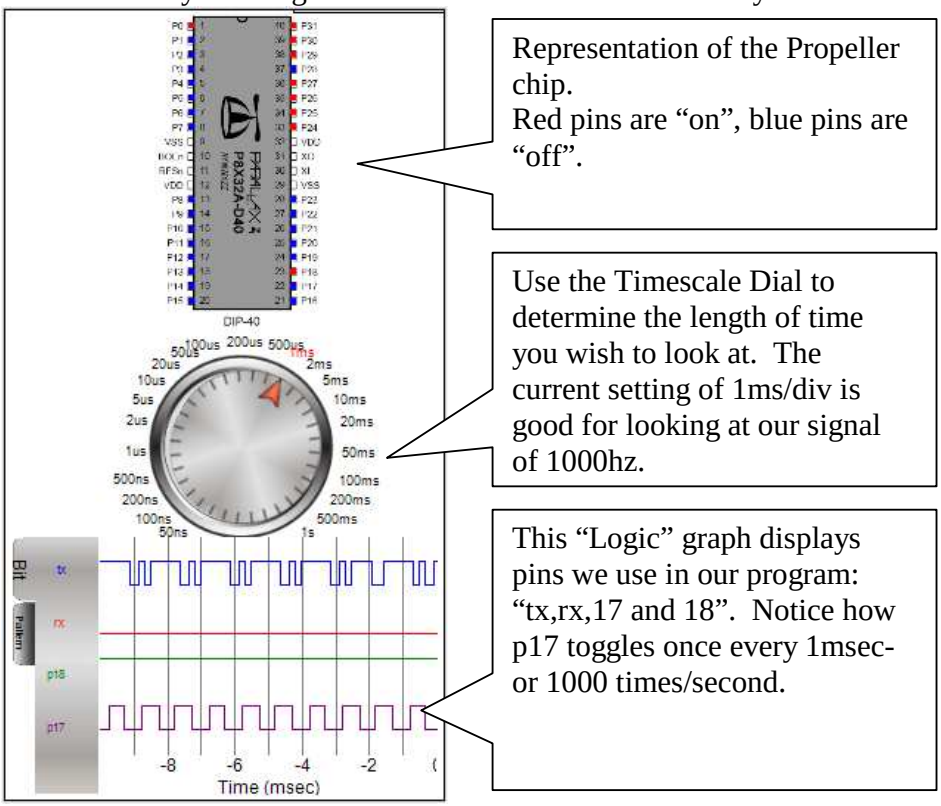
Blinking one light is easy, now blink another one. Drag another *start* block to the worksheet and assemble your program so it looks like this:



Instead of using *wait* and *toggle* to blink the LED, we're using the *output frequency* block. This let's us easily output any frequency up to tens of Megahertz on a pin. By using two start blocks, we're using two "cogs" and running our program in parallel. This is a very powerful technique that makes the Propeller ideal for creating powerful programs- and 12Blocks makes it very easy to use.

Press the *Run* button to see the two LED's blink.

Now that your program is running, change the frequency from `11` to `5` by clicking on the yellow parameter field, typing 5 and pressing enter. You won't need to stop and recompile, 12Blocks let's you make changes to parameters with numbers while your program is running! You should see the LED blink more slowly. Now, change the frequency to `1000`, a speed that's too fast for your eyes to see. The LED will appear to be on all the time, so we need a tool to make sure the LED is actually blinking. Press the "View Pins" button and you should see this view:

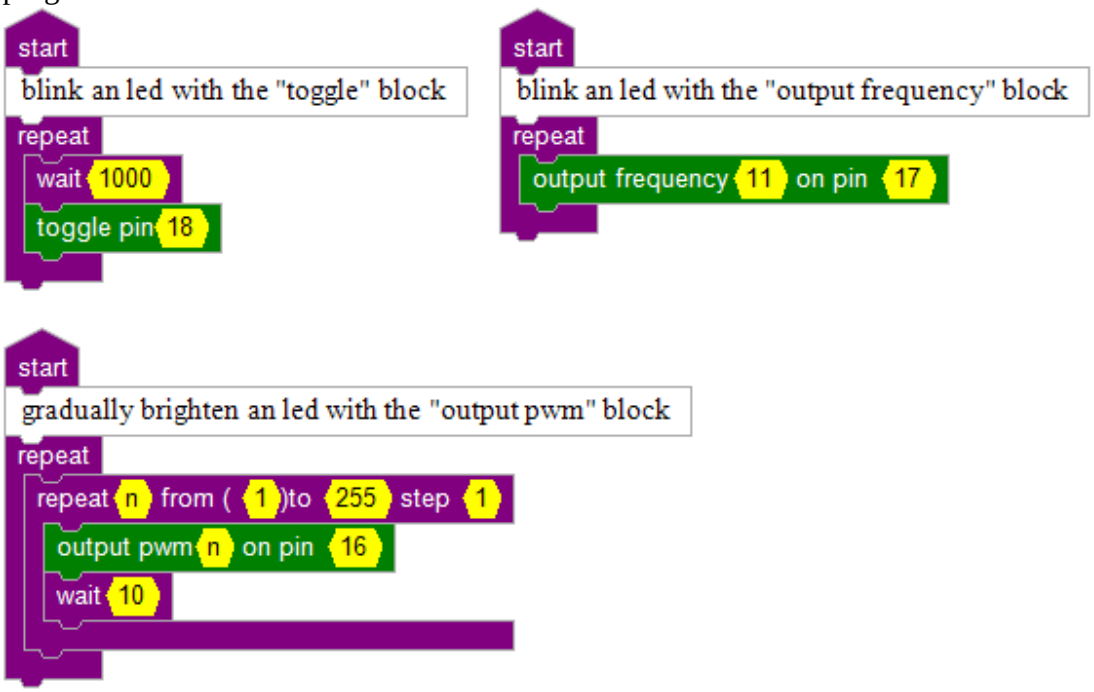


Representation of the Propeller chip.
Red pins are "on", blue pins are "off".

Use the Timescale Dial to determine the length of time you wish to look at. The current setting of 1ms/div is good for looking at our signal of 1000hz.

This "Logic" graph displays pins we use in our program: "tx,rx,17 and 18". Notice how p17 toggles once every 1msec- or 1000 times/second.

Finally add a start block and blink a third LED with pulse width modulation. Assemble your program so it looks like this:



Notice that we're using a different *loop*- one that let's us specify a *variable* and counts from a *start* to a *stop* value with a given *step*. In our case, the loop will count `n` from `1` to `255` in steps of `1` and then repeat.

Also notice the *output pwm* block. This very quickly toggles it's pin- but stays *on* for different amounts of time. When *n* is `1`, it will only turn on for a very short time, which makes the LED appear to be *off*. When *n* is *255*, it will turn on for a very long time, making it appear *on*. At *128*, the LED will be *on* as long as it's *off*- so it will appear "dim".

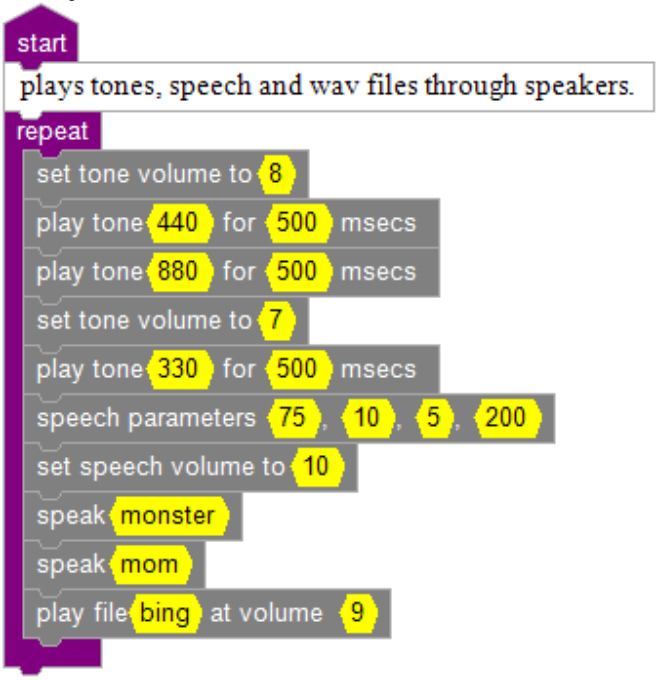
The *wait* block determines the frequency of the blinking. As configured, it will do one cycle in 255 steps*10ms/step=2.55seconds.

Run this program and you should see pin 16 repeatedly getting brighter.
While the program is running, change the start value from a `1` to `100`. This will brighten the LED from *dim* to *on*. Finally change the *stop* value from `255` to `0`. This will gradually darken the LED from *dim* to *off*.

Program #3: Sounds

Connecting speakers/microphone to a Propeller:
The Demo Board includes a built-in microphone and a socket for headphone speakers. Refer to your Propeller documentation to build the simple circuits if you're using different hardware.

It's easy and fun to create sounds with the Propeller and 12Blocks. This program (`sounds.12b` in your *Tutorial* folder) demonstrates some blocks from the *audio* section of the library:



Here's what this program does:

- It starts with one *start* block and a *comment*
- It uses a *repeat* block to continually run blocks from the *audio* section of the library
- The *tone volume* is set to 8 and two tones are played for 500 msecs each. A tone of 440hertz is called *Concert A*, on the piano it's the first A to the right of Middle C. Playing a tone at twice that frequency makes it an octave higher.
- The *tone volume* is then set to 7 and another lower note is played.
- The *speech parameter* block configures how quickly and with what qualities words are spoken by the Propeller. The *speech volume* block changes the volume.
- The *speak* blocks cause the Propeller to speak words. Click on the parameter to change the word.
- Finally, a WAV file is played.

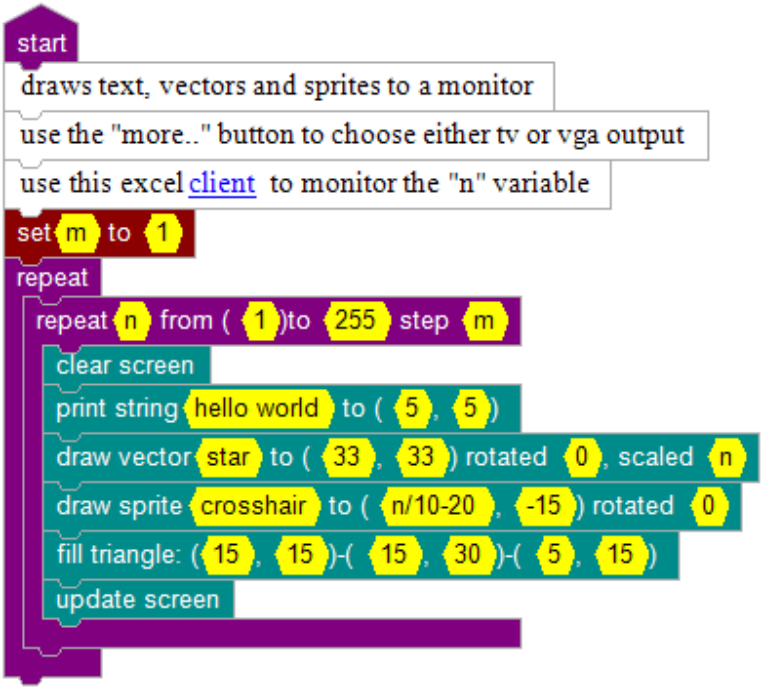
Additional things to try:

- Click *More..* and then select *Save As* to save the file with a new name
- Click *More..* and then select *Print* to print your code.
- Click *More..* and then select *Report Bug* to report a bug.
- Use other *Audio* blocks to record and play back sound

Program #4: Graphics

Connecting your Monitor/TV to a Propeller:
The Demo Board includes a VGA connector for use with most computer monitors as well as a composite socket to connect to most TV's. Refer to your Propeller documentation to build the simple resistor circuit if you're using different hardware.
12Blocks supports graphics and text output to both VGA and TV monitors. Click the *More..* button and select which device you want to use.

You've seen how easy it is to assemble some audio blocks to play music with the Propeller, now give graphics a try! Assemble this program from the blocks in the library's *graphics* section (or load `graphics.12b` in your *Tutorial* folder)



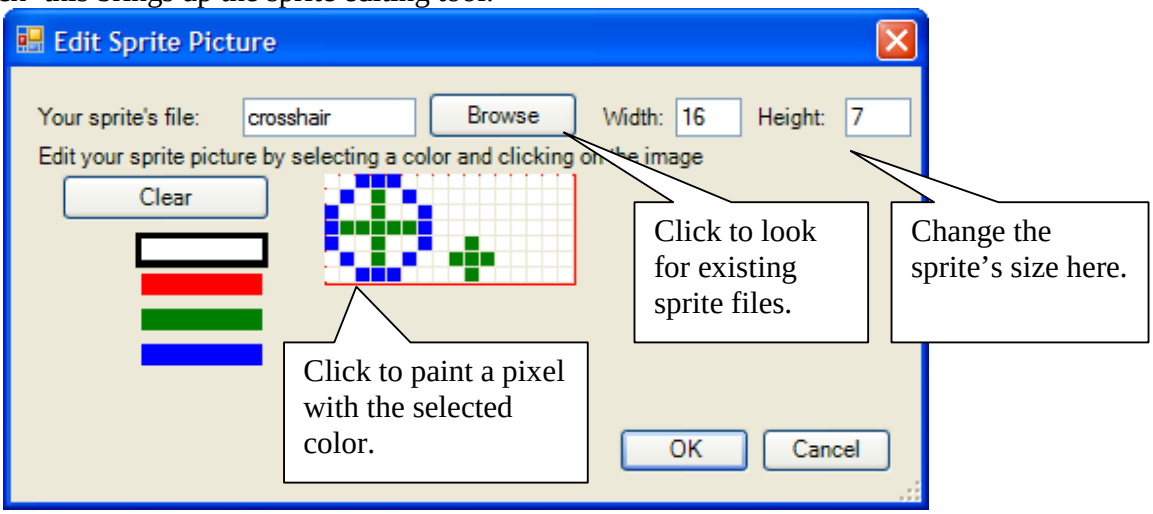
Run your program and you should see text, a vector star, a crosshair sprite, a triangle on your screen. The items will move and change in size. Here's what the program does:

- The program uses 1 *start* block
- The *comment* blocks tells us a bit about the program
- The red *set* block sets the global variable *m* to a value of *1*. Global variables are defined just by using them. Later we'll see how to change this variable from other programs like Excel.
- The outer *repeat* block continually runs the inner *repeat.*
- The inner *repeat* causes variable *n* to count from *1* to *255* with step *m.* Variables have names that start with letters and can store integer values, like -1, 10, or 2,000,000. To use a variable, just type its name into the parameter field of a block, or use it in an expression.
- Finally we get to the real meat of our program, with blocks from the *graphics* section of the library. At the start of each cycle, it *clears the screen*
- Then *hello world* is printed at position *(5,5)*

- A vector named *star* is drawn to (33,33) and scaled by *n.* See below to learn what a vector drawing is and how to edit one. The scale factor means that the drawing will get larger as n increases.
- A sprite named *crosshair* is drawn to *(n/10-20,-15).* The expression *n/10-20* first divides *n* by 10 and then subtracts 20, so, as *n* gets larger, the sprite will slowly move further to the right. See below to learn what a sprite is and how to edit one.
- A filled triangle is drawn at specified vertices.
- Finally, the program *updates the screen* with the new graphics- without block, you wouldn't see your graphics, so don't forget about this block!

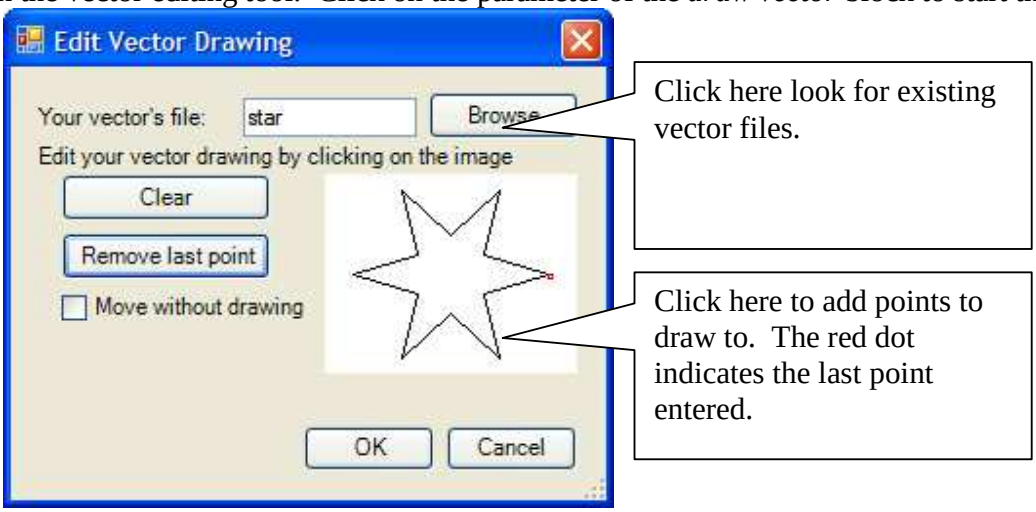
Drawing with Sprites

Sprites are like stickers, they can contain intricate drawing in multiple colors and are easily placed where you want them. You edit a sprite by clicking on the parameter of the *draw sprite* block- this brings up the sprite editing tool:



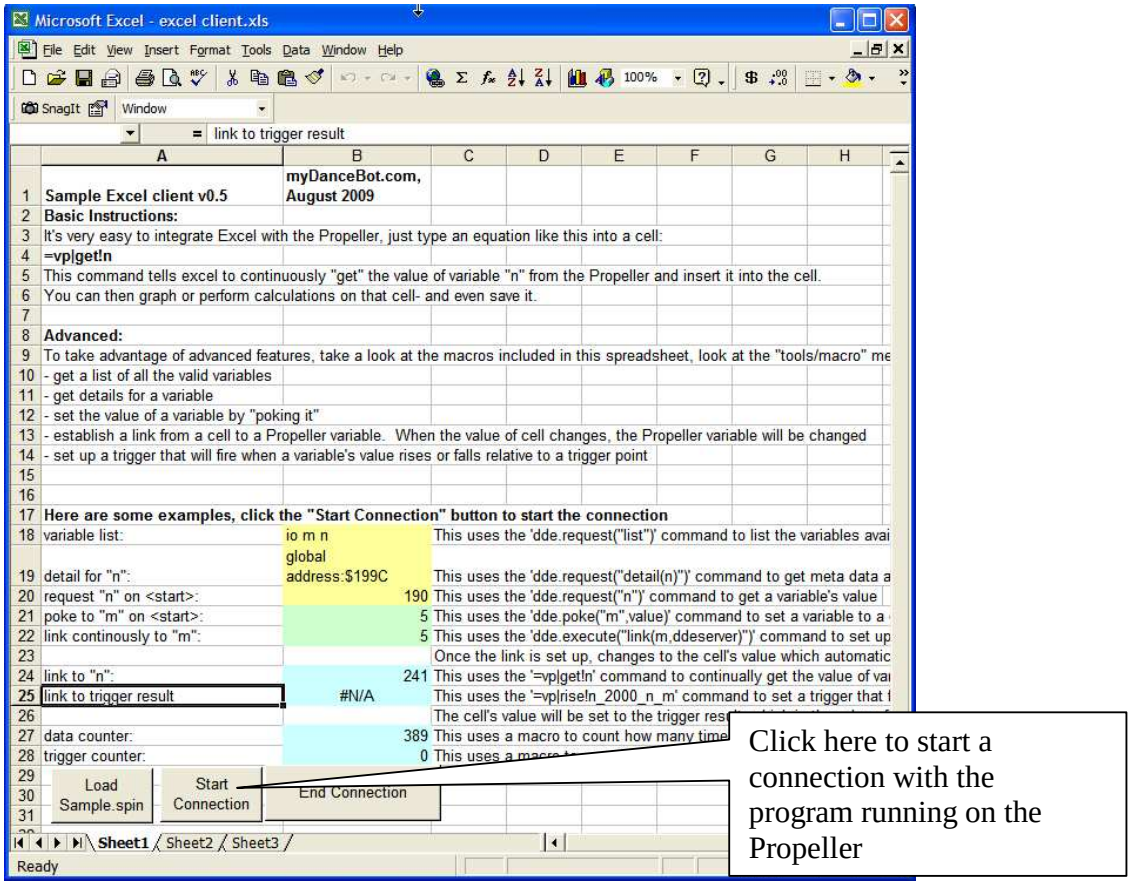
Drawing with Vectors

Vectors are like "connect the dot drawings", lines are drawn between points which you can edit with the vector editing tool. Click on the parameter of the *draw vector* block to start the tool:



If you've made changes to parameters like a text string, an expressions, a sprite or a vector, you'll need to click *Run* to load the modified program to the Propeller to see your changes. If just changed a parameter from one number to another you'll see the change right away- like moving an object or rotating it.

This program has two variables named *n* and *m*. The variable *m* defines the step size with which *n* increases- it's set to 1 initially, which means *n* counts like this: 1,2,3,...
If we set *m* to 2, *n* will count by 2's- and so the animation will go twice as fast. You've seen how to make changes to the program running on the Propeller from within 12Blocks, now you'll make changes to a program running on the Propeller using Microsoft Excel. The last comment links to a *excel client*- click it to open the excel spreadsheet on your computer. The spreadsheet uses macros so you'll need to enable them when prompted. It looks like this:



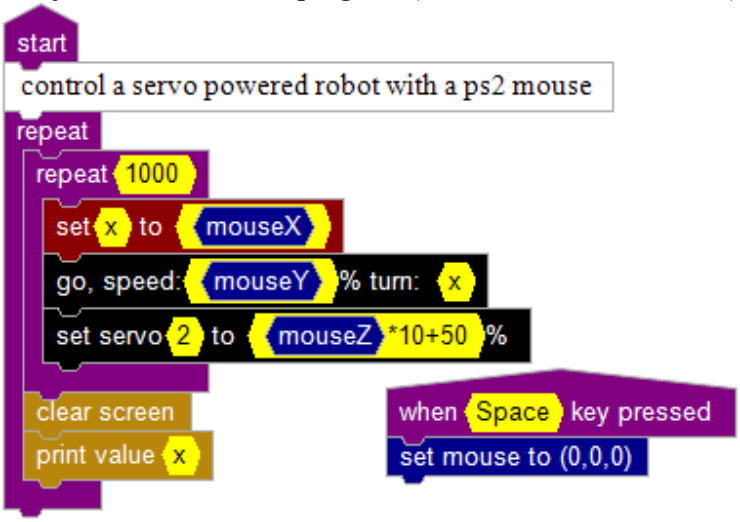
Once you click on the "Start Connection" button in the spreadsheet, you'll see cell *B24* change- it continuously shows the current value of the variable *n* from your program. Cell *b22* is linked to variable *m*, when you change this to 2, the graphic animation will go twice as fast. Read the rest of the spreadsheet to learn how to control and monitor global variables in your Propeller programs with 12Blocks.

Integrating with other languages/applications:
To learn how to integrate with other applications and languages like C#, VB.NET, Python, Delphi and Matlab, visit our integration website: http://12blocks/integrate.php

Program #5: Controlling servos with Mouse and Keyboard

Connecting servos, mouse and keyboard to a Propeller:
The Demo Board includes sockets for a PS/2 Mouse and Keyboard. Refer to your Propeller documentation to build the simple circuits if you're using different hardware. To connect a hobby servo to your Propeller, first plug the servo into a breadboard. Then, use wires to connect the servo's black wire to ground, labeled VSS. Connect the servo's red wire to 5V. And connect the servo's white wire to one of the Propeller's pins with a 100ohm resistor.

Time to build a simple robot- or at least figure out how to control hobby servos with a mouse and a keyboard. Here's the program (from *Tutorials/robot.12b)*



To use it, confirm you've connected a mouse and keyboard to your Propeller and servos on pins 0, 1 and 2. When you run the program two motors should follow the (x,y) position of your mouse, while the mouse's scroll wheel controls the third. Pressing the *space* key on your keyboard should temporarily bring the servos to their center position.

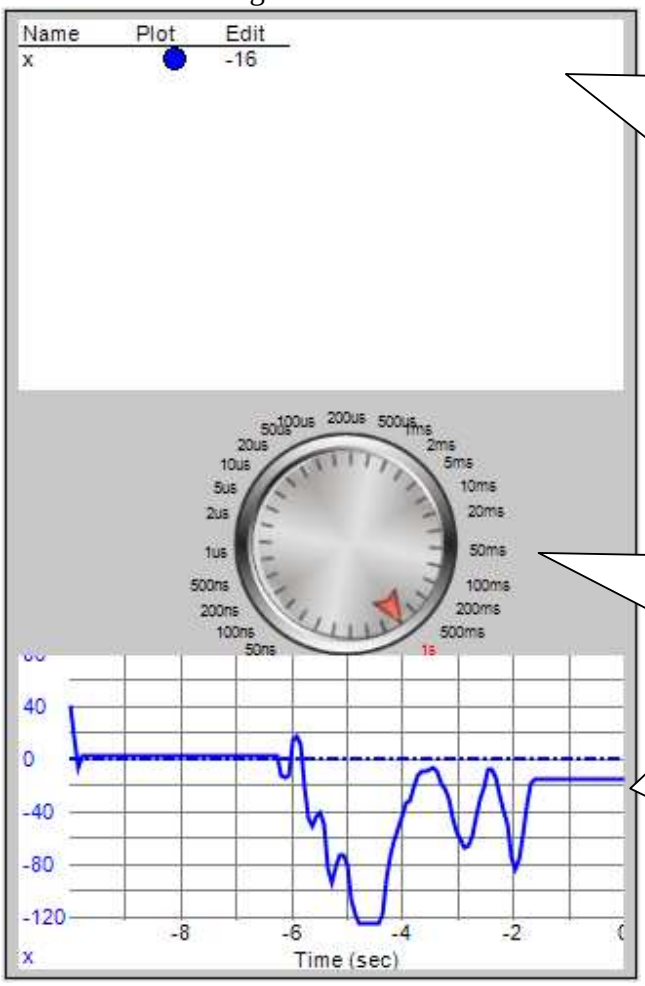
Let's see what the program does:

- It uses 2 *start* blocks.
- The lower one will only run when the *space* key on the keyboard is pressed. It handles an event- other even handlers are available to detect if the mouse button has been clicked, if a character has been typed into the terminal, or when a message has been sent- more about that later. This handler just sets the mouse position to 0.
- The upper *start* block starts with a comment and then continually loops.
- Another *repeat* block loops 1000 times to control the servos before updating the terminal screen.
- It sets the global variable *x* to *mousex*, a result block which returns the horizontal location of the mouse. Notice that most blocks can be dragged into the yellow parameter region of another block. The only blocks that can't be parameters are blocks like "repeat" and "start"- that wouldn't make sense. Some blocks, like *mousex* can only by placed into a parameter region- that's why it doesn't have any tabs.
- The *go* block from the black *motion* library section allows you to control the *speed* and *turn* rate of a robot driven by two continuous rotation robots with one command. There are also

blocks to make your robot go *forwards*, *backwards*, and *turn*. We pass it two parameters, the block *mousey*, and the variable *x*. This allows you to control the robot by moving the mouse up/down to control it's speed and left/right to control it's direction.

- The *set servo* block sets the position of a servo given it's pin number and target position given in percentage- 0..100%. Our expression combines the *mousez* block with a formula.
- After the *repeat* block loops 1000 times, the terminal screen is cleared and the value of variable *x* is printed to the terminal.

When you run the program you can click the *View Terminal* button to see the value of *x* change as you move the mouse. 12Blocks also let's you view the value of all global variables and graph them over time using the *View Values* button.



This area shows us a list of all global variables. If you wish to plot a variable, click the circle- it's color shows you which graph trace corresponds to which variable. The number in the *Edit* column indicate the current value of the variable. Click and change it if you want to!

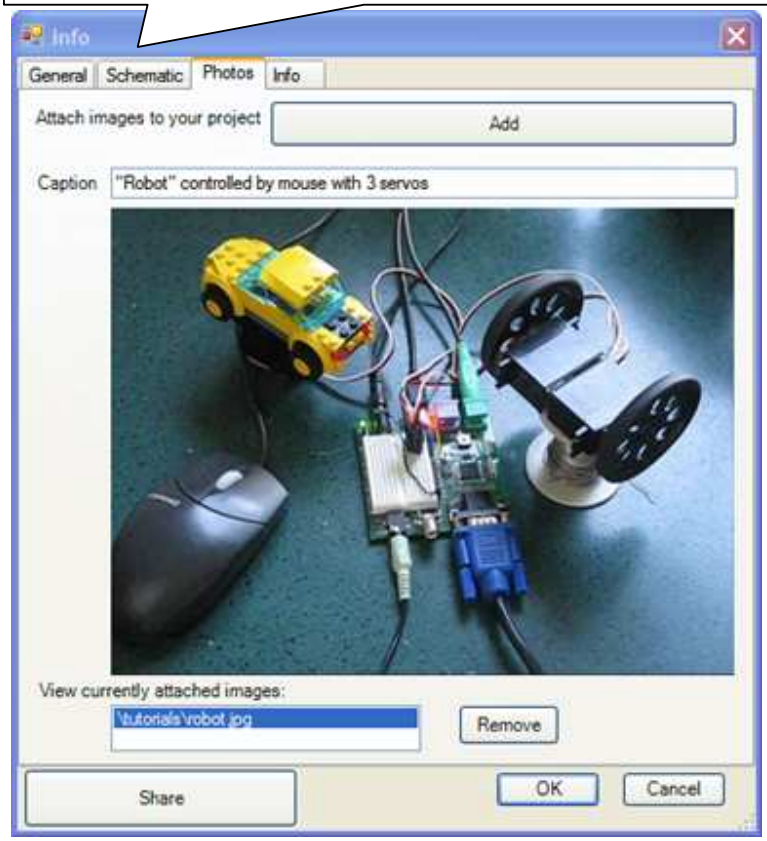
Use the Timescale dial to determine the length of time you wish to look at. The current setting of 1s/div is good for looking at our signal which changes very slowly.

This graph shows the values of one or more variables over time. You can drag the trace up and down to move it, or left/right to go forwards or backwards in time. Click to the left of the graph to set a trigger.

Sharing your Program

12Blocks makes it easy to share your creation with the world. The *Info* button let's you view and change information about your program. You can attach images to your program and edit schematics for your circuits.

Add general information like a description and name, edit a schematic, or attach photos and other images.

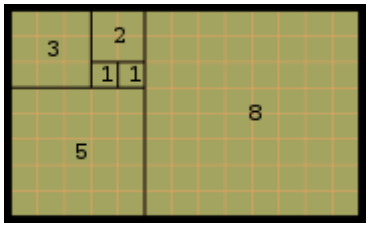


Click the *Share* button to upload the entire project to the searchable online database: http://12blocks.com/howto

Additional things to try:

- Click *More..* and then select *Load to EEPROM* to load the program into the Propeller's EEPROM. This lets you use your robot away from your PC. Your program will restart running at the beginning whenever the Propeller is reset.
- Click *More..* and then select *Copy Code to Clipboard* to copy an image of the code to the clipboard.
- Click *More..* and then select *Copy View to Clipboard* to copy an image of the view to the clipboard.

Program #6: Calculating Fibonacci Numbers with Functions



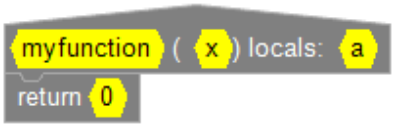
You might have heard about the Fibonacci sequence in movies like "The DaVinci Code". Let's calculate some of the numbers with 12Blocks!

Here are the first numbers in this sequence- notice a pattern?
0,1,1,2,3,5,8,13…

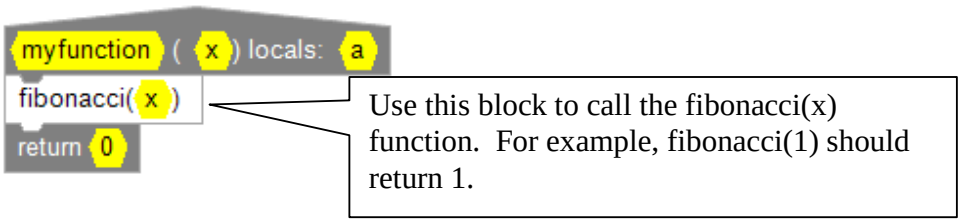
The first two numbers are 0 and 1, and after that the fibonacci number is the sum of two preceding fibonacci numbers. Mathematically, we can define that like this:

Fib(0)=0
Fib(1)=1
Fib(x)=Fib(x-1)+Fib(x-2)

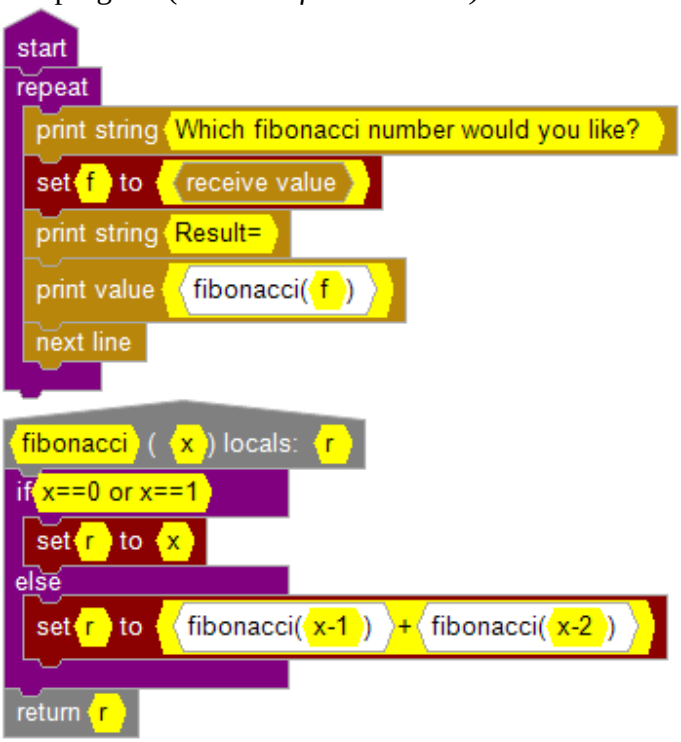
To calculate this with 12Blocks, we need to explore the *functions* part of the library. This section initially only contains two blocks.



When you drag the *myfunction start* block to the worksheet, you should replace the `myfunction` parameter with a name of your function. You can also replace the argument parameter with a comma separated list and name your local variables. This will create a new block in the *functions* section of the library- which you can use to call your new function:



Now that you know how to create functions and how to call them, you should be able to create this program (*Tutorials/fibonacci.12b)*:



Besides using a function, this program also uses the *if else* block. This block has two internal attach tabs. ***If*** the *condition* is true, in our case `x==0 or x==1` then the first part is used and we set `r to x.` ***Else*** we set *r* to the sum of *fibonacci(x-1)* and fibonacci(*x-2)*.
Finally, our function returns the local value of local variable *r*.

This program uses the terminal to prompt you for which fibonacci number you wish to calculate. It then calls the fibonacci function which returns right away if x was set to 0 or 1. Otherwise it will call the fibonacci function for (x-1) and (x-2). This is called *recursion*.

Recursion is sometimes the most straightforward way of programming functions, but it's often not very efficient and doesn't work well for large numbers- we'll fix that with the next program.

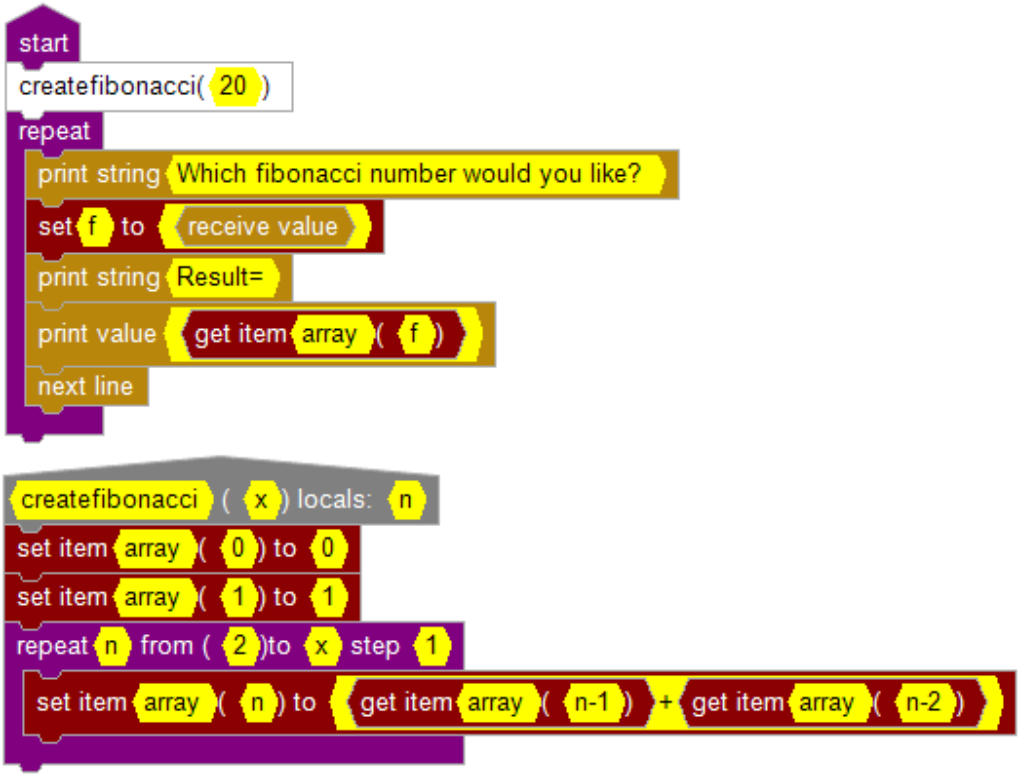
Program #7: Calculating Fibonacci Numbers with an Array

Now that you know how to calculate Fibonacci numbers with a recursive function, try a different way of calculating the sequence. Instead of calculating the sequence when prompted, store the sequence in the Propeller's memory.

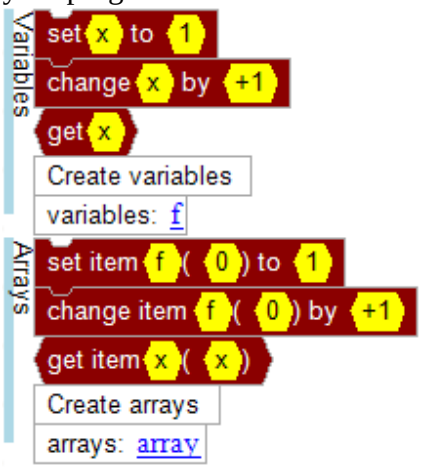
You already know about variables which are used to store the value of a single element. Arrays can store multiple elements- ideal for storing a sequence of Fibonacci numbers. Once we've stored the Fibonacci numbers in the array, retrieving the *n*th Fibonacci is as easy as using the *get item* block from the *array* group of the *vars* section of the library.

The first part of the following program (*Tutorials/fib array.12b*) creates the Fibonacci sequence and then prompts the user and displays the item from the array.

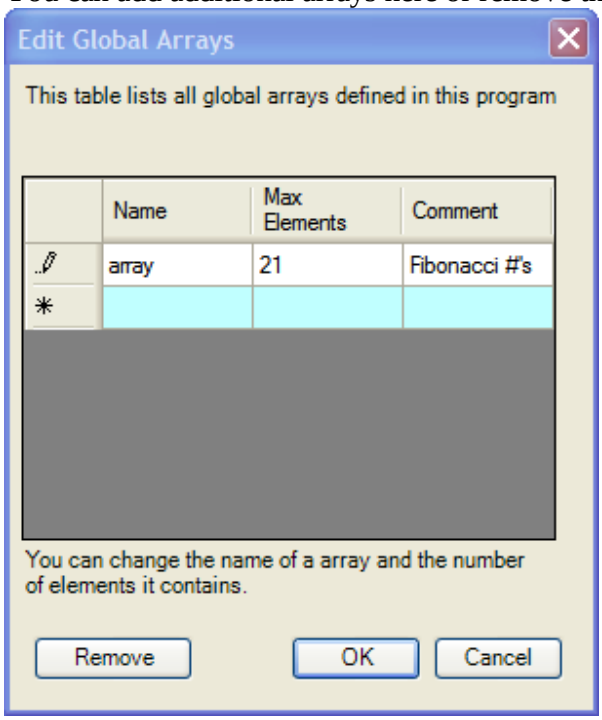
The second part calculates the sequence by first setting the first two entries of the array and then looping from the second item to the *xth* to set them. It sets the *nth* element by summing the *n-1* and the *n-2th* elements.



The *vars* section of the library contains three groups of blocks, one for *Variables* another for *Arrays* and another for *Interface blocks.* There are blocks to *set, change, get, and edit* an array or variable item. The *Interface* group contains a block that displays variables and arrays used in your program.



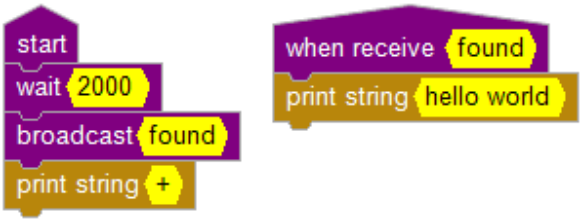
The *Edit Global Array* tool shows all global arrays with their maximum elements and comments. You can add additional arrays here or remove them.



Program #8: Broadcasting Messages

You've seen how to handle mouse, keyboard and terminal events by using the appropriate *when start* blocks. For example, this program will print `Hello World` when the mouse is clicked:

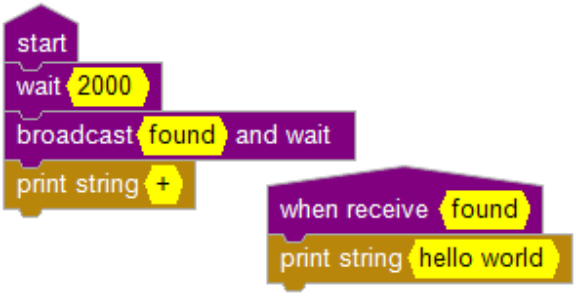
In addition to handling mouse, keyboard and terminal events, 12Blocks also let's you *broadcast messages* and handle them in a *when receive* block. This program will wait 2 seconds and then broadcast a message called *found* and immediately continue to the next block where + is written to the terminal. The *found message* starts a new cog and causes `hello world` to be printed.



The result of this program is:

```
+hello world
```

In this case, the message was processed asynchronously. You can use the *broadcast and wait* block to do things synchronously- meaning one after the other. What do you think this program will do?



That's right, it *waits* for 2 seconds and then *broadcasts* the *found* message which will be handled and print out `hello world.` When the handler is finished, the next block will print a +, so the output of this program is:

```
hello world+
```

Additional things to try:

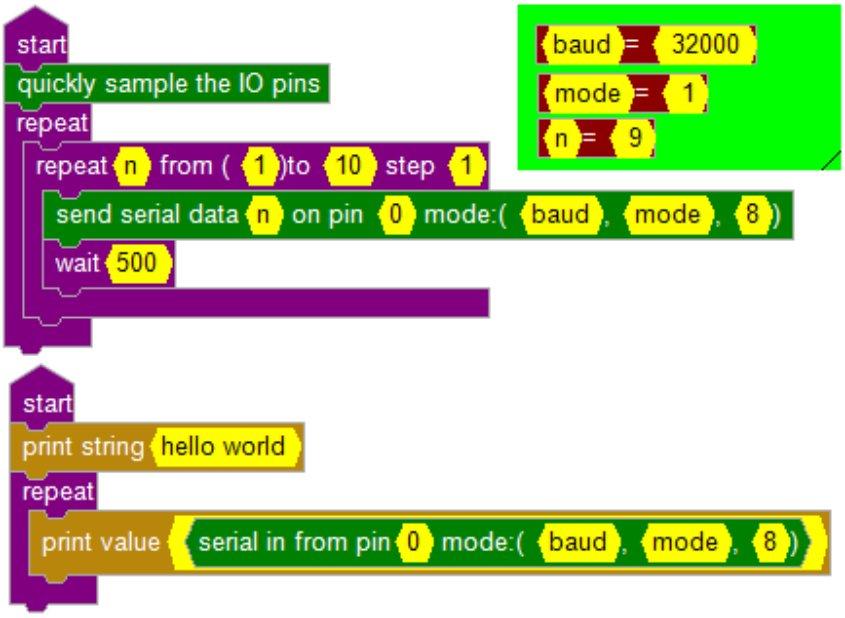
- Click the *Add Block* button to start a wizard to add a block to your library. This let's you add additional functionality to 12Blocks.
- Click *More..* and select *View Code* to view the spin code for your program in the Parallax Propeller Tool.
- Click *More..* and select *Hardware*- click to select from different hardware targets. A target file specifies the clock rate, pin layout, and experience level.
- Investigate the "*Tutorials/Events.12b" program.*

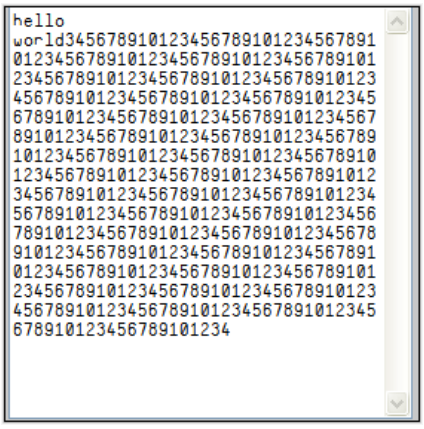
Program #9: Serial Communication

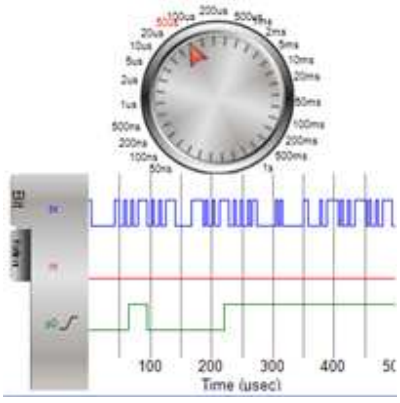
The following program *(Tutorial/Serial.12b)* uses the serial communication blocks from the *pins* section of the library to send information from one cog to another. The top stack tells 12Blocks to quickly sample the IO pins, this let's us view the resulting RS232 signal with the built-in logic analyzer at high speeds. The inner repeat loop sends one byte of data and then waits 500mSecs. Both the baud rate and mode are configurable. A mode of 1 will leave the pin high by default.

The bottom stack runs in another cog. It first prints a friendly greeting to 12Blocks' terminal and then continually prints incoming data.

The green rectangle and its contents are "*user interface*" blocks found in the *vars* section of the library. Use these blocks to monitor and change variable values directly from the worksheet. When your program is running the values will update in real time. You can also click the value and change it- try changing the baud rate to 9600.







Program #10: State Machines

A state machine is a model of behavior with states and transitions between states. For example, a door has two states- open and closed. It also has two transitions- opening and closing. State machines can make it easier to program sophisticated behavior by letting you focus on what happens in one state at a time.

12Blocks includes 3 blocks to make state machines easy to use:

- "*when in state"* Use this block as a start block to carry out actions that should be performed when the state machine is in a specified state.
- "*set state to":* Use this block to transition to a new state specified by the parameter.
- "*run state machine":* Use this block to run one cycle of the state machine. The variable which you pass to this block is used to keep track of the state machine's state. By using multiple "*run"* blocks, each with their own variable, you can program multiple state machines.

The following program *(Tutorials/StateMachine.12b)* illustrates a simple state machine.