# Get Started with the Propeller Board of Education

If you just got your Propeller Board of Education and you are new to Spin programming, this is the place to start!

Beginner-friendly objects are provided for every activity — just download the code and go!
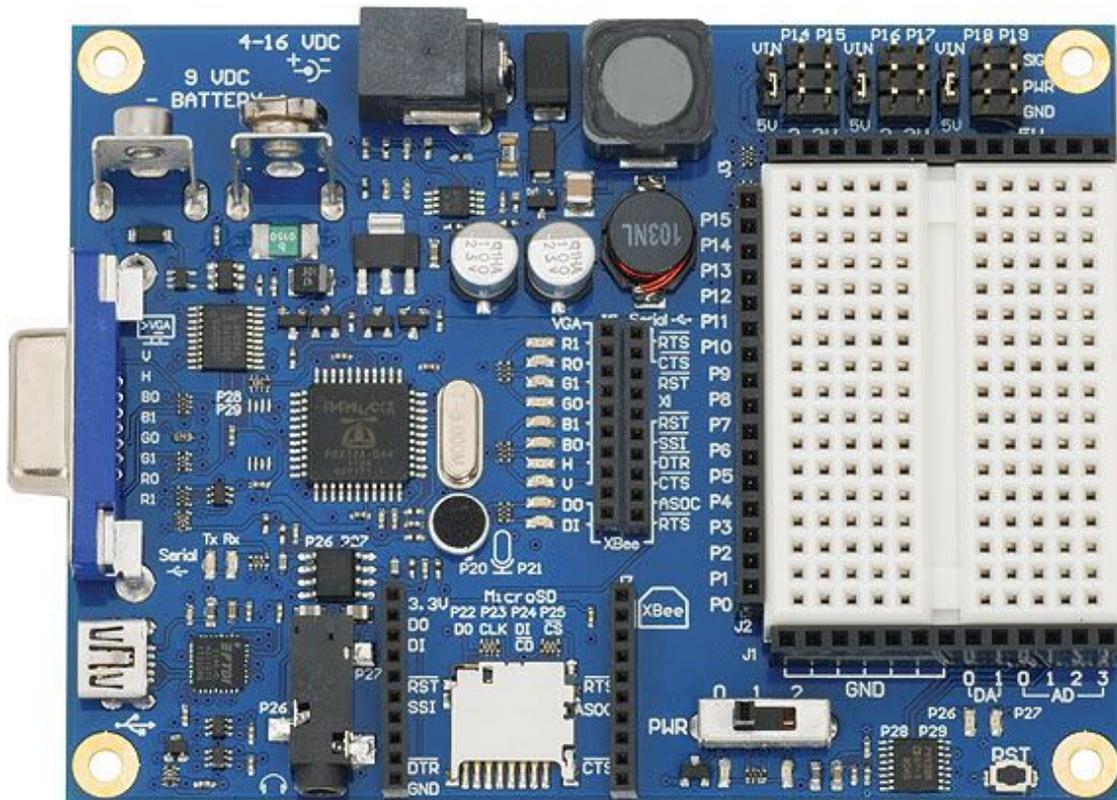
Lesson 1: Install the Software

This section will guide you through installing and configuring the Propeller Tool software.

Download the Software

- ✓ You will need a computer with Windows 2K or newer and Internet Explorer 7 or newer.
- ✓ [Download the latest version of the software.](from Parallax.com) (from Parallax.com)

Install the Propeller Tool

- ✓ Run the installer you just downloaded, and accept all default settings.

Important! On the Install Optional Driver step, make sure to leave the checkbox checked for the "Automatically install/update driver" feature.
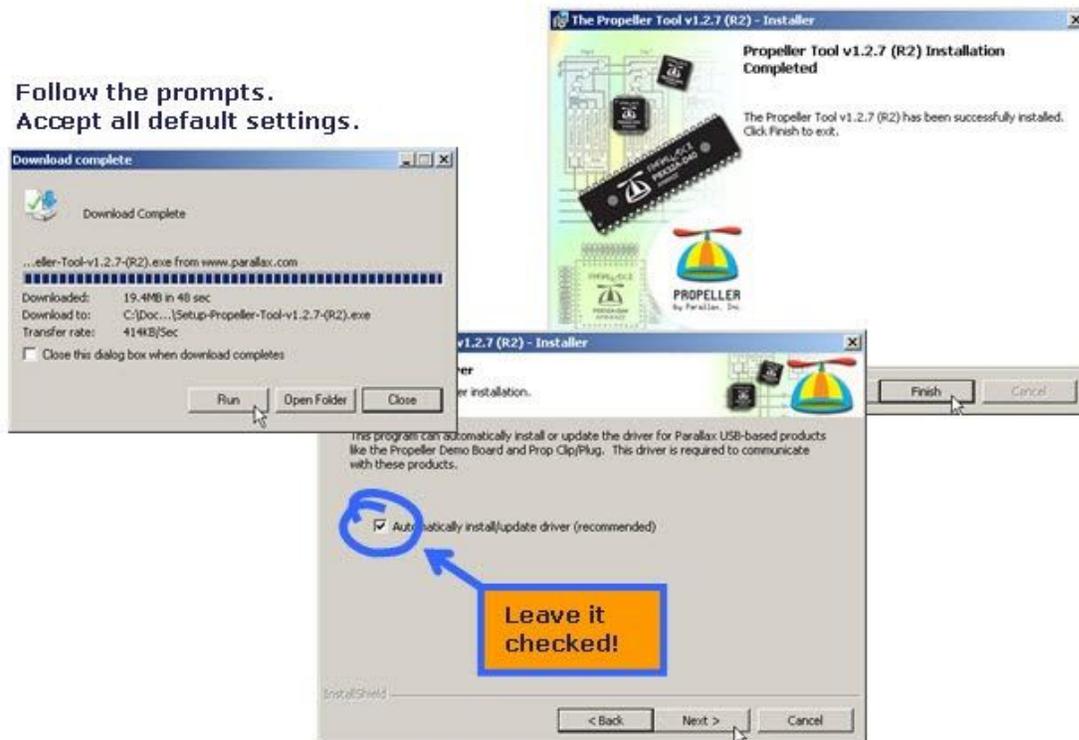

Run the Software

- ✓ When you double-click the Propeller Tool software link that the installer placed on your desktop for the first time, it will ask you about file associations. Click Yes.
- ✓ Double-click the Parallax Serial Terminal icon to open it too.
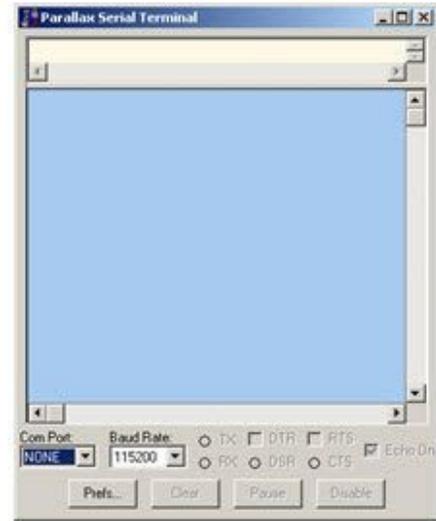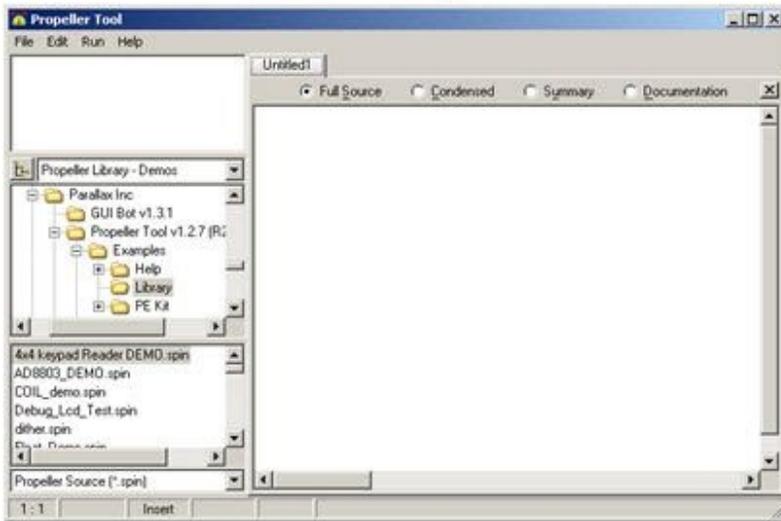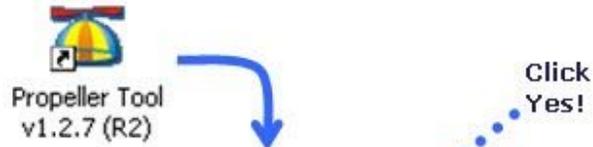
Connect the Hardware

- ✓ Connect your Propeller BOE to your PC's USB port.
- ✓ Wait while Windows sets up the USB driver.
- ✓ Set the 3-position switch to Position-1 and check for a green light.
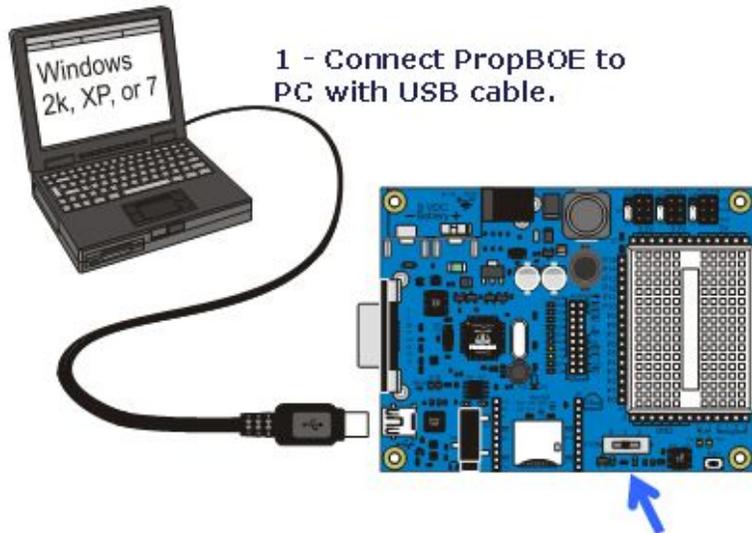
1 - Connect PropBOE to PC with USB cable.

2 – Wait for USB COM port.

3 – PWR switch to position 1.
Check for one green light below switch.

Configure the Software

✓ Click the Run menu and select Identify Hardware to find out what COM port your Propeller BOE is connected to.

✓ Set your Parallax Serial Terminal with your Propeller BOE's COM port.
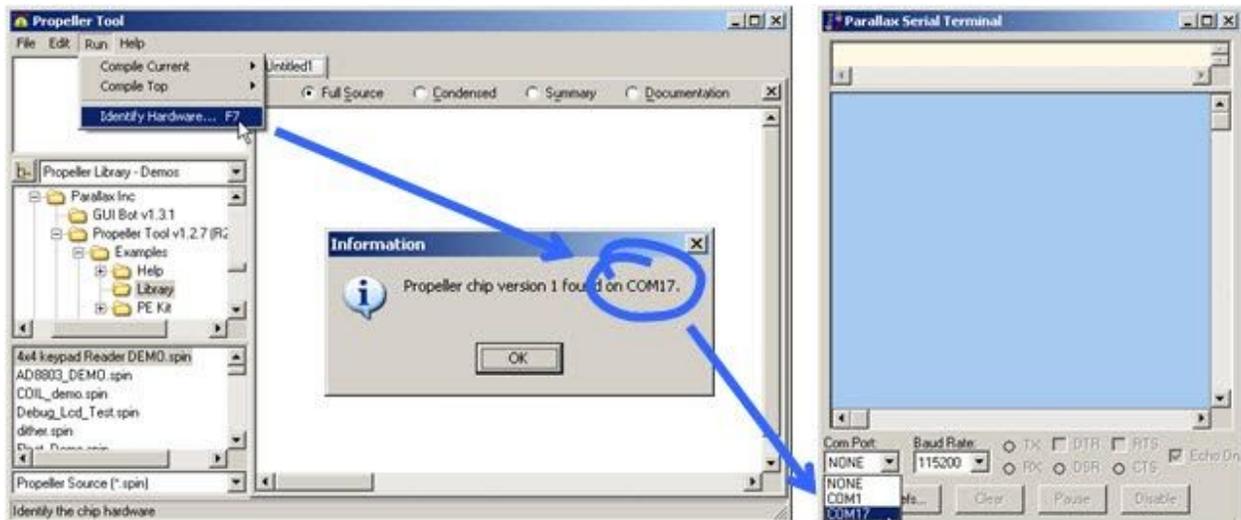
Lesson 2: LED Light Control

The Propeller microcontroller has 32 input/output pins. It can be programmed to set these I/O pins to certain output voltages and to monitor for certain input voltages. In this lesson, we'll connect some of these I/O pins to light emitting diodes (LEDs) that are built into the Propeller Board of Education. Then, we'll use Spin programs to control the Propeller I/O pin voltages, and so make the LEDs blink on and off.

  ✓ Open the LED Light control subfolder (remember to always open programs from within an unzipped folder).
 ✓ Follow the sections below to continue with the lesson.

Blink a Light

The Propeller microcontroller has 32 input/output pins, abbreviated "I/O" pins. The Propeller chip can be programmed to set these I/O pins to certain output voltages and monitor for certain input voltages. (Voltage is abbreviated V.) In this first activity, we'll connect a light emitting diode circuit (abbreviated LED) to an I/O pin, and then write a program that makes the Propeller chip alternately supply 3.3 V and 0 V to the I/O pin. The result will be that the light in the LED circuit will alternately turn on and off.

Build a Test Circuit

It takes one wire to connect a Propeller I/O pin to one of the PropBOE's built-in LED circuits.

  ● Take a jumper wire and plug one end into the socket labeled P9, and plug the other end into the socket labeled H.

The blue fiberglass printed circuit board has a built-in connection between the P9 socket and the Propeller microcontroller's P9 I/O pin.
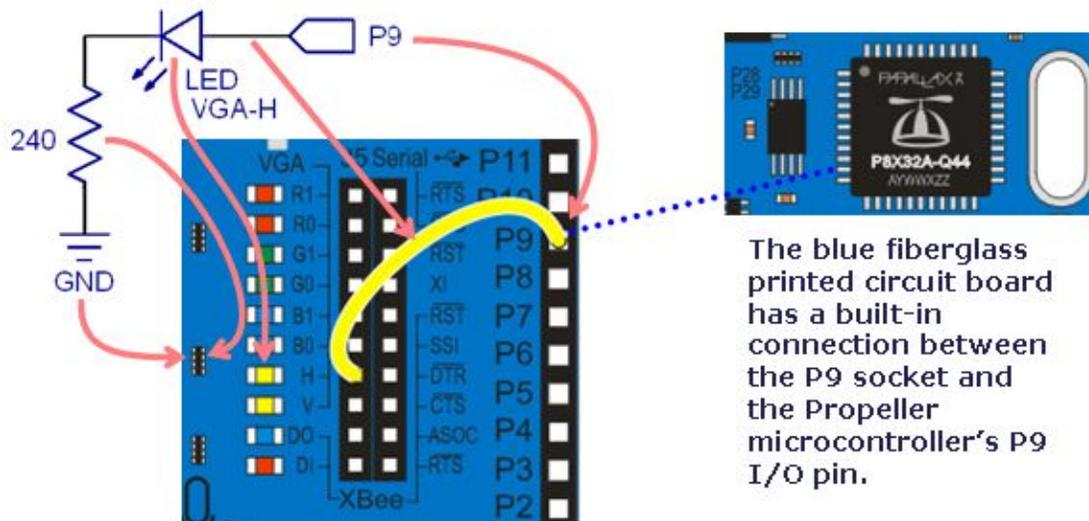
The figure shows how the components in the circuit schematic relate to the PropBOE. The LED is the tiny component just to the left of the H label, and to the left of the LED is a resistor with a

value of 240 Ω. The ohm is a measure of how strongly the device resists current, and it's abbreviated with the Greek letter omega Ω.

Before writing and modifying programs, let's try a simple test program and make sure the light blinks.

✓ Open the folder "PropBOE_Spin_Leds_20120130".

Open and Run First Example Program

✓ Find and double-click 1 Blink Light.spin. It should automatically open into your Propeller Tool software.
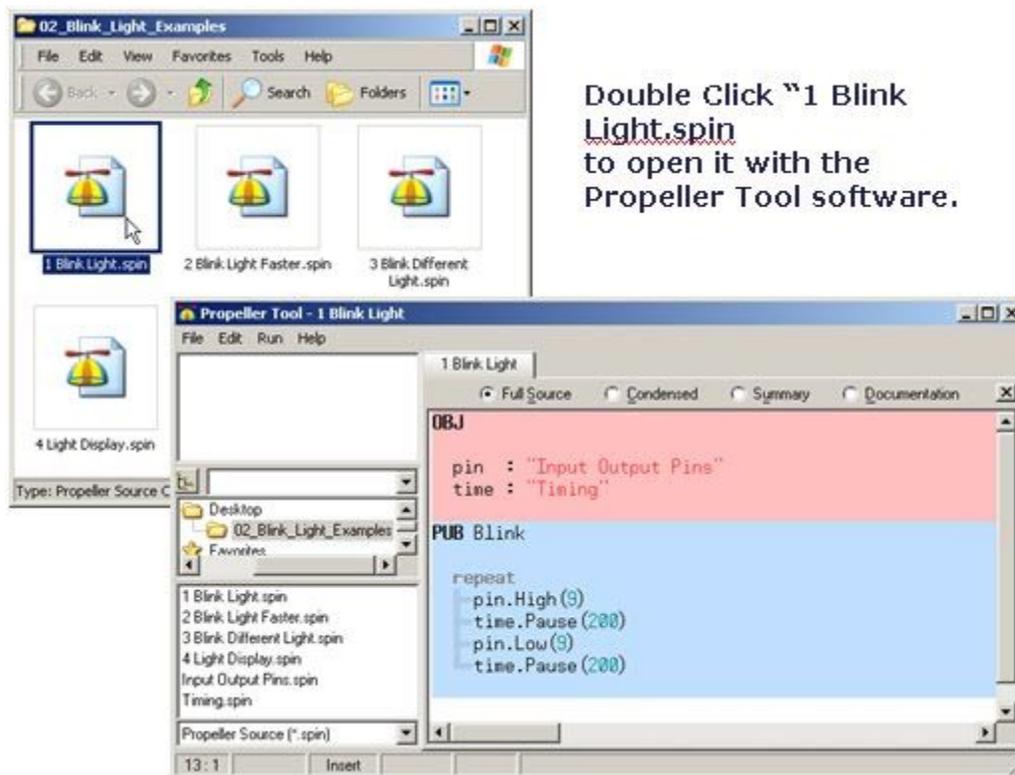
✓ Click Run, then point at Compile Current.
✓ In the Compile Current submenu, click Load EEPROM. (F11 is the shortcut for loading your program into EEPROM.)
✓ Check your board for a yellow blinking light.
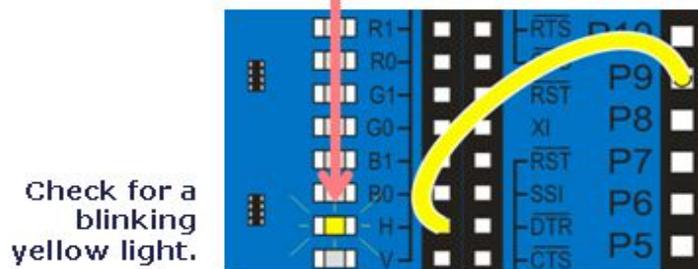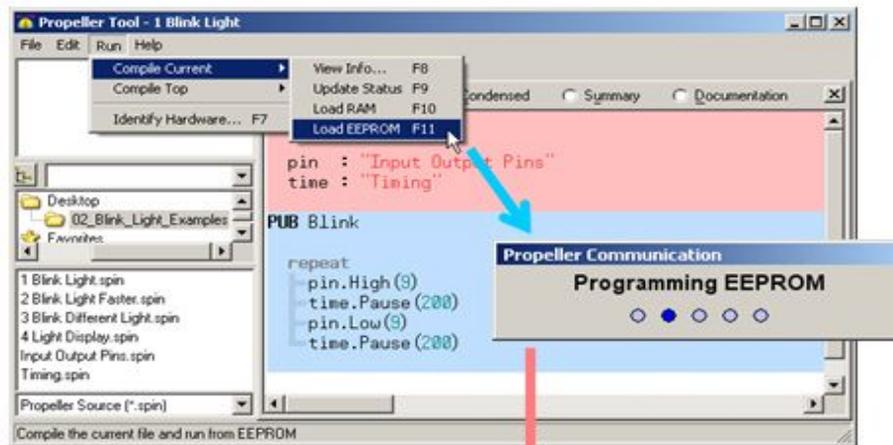
How the Blinking Light Program Works

Spin Objects

Files that contain Spin language are called *objects*. Our example program is in a file named "1 Blink Light.spin". This object uses code in two other objects, one named "Input Output Pins.spin" and another named "Timing.spin". These are called *user library* objects, and they contain blocks of code called *methods* that your program can use, almost like Spin commands. It's kind of like having files that give the Spin language custom "commands" to get any type of job done.
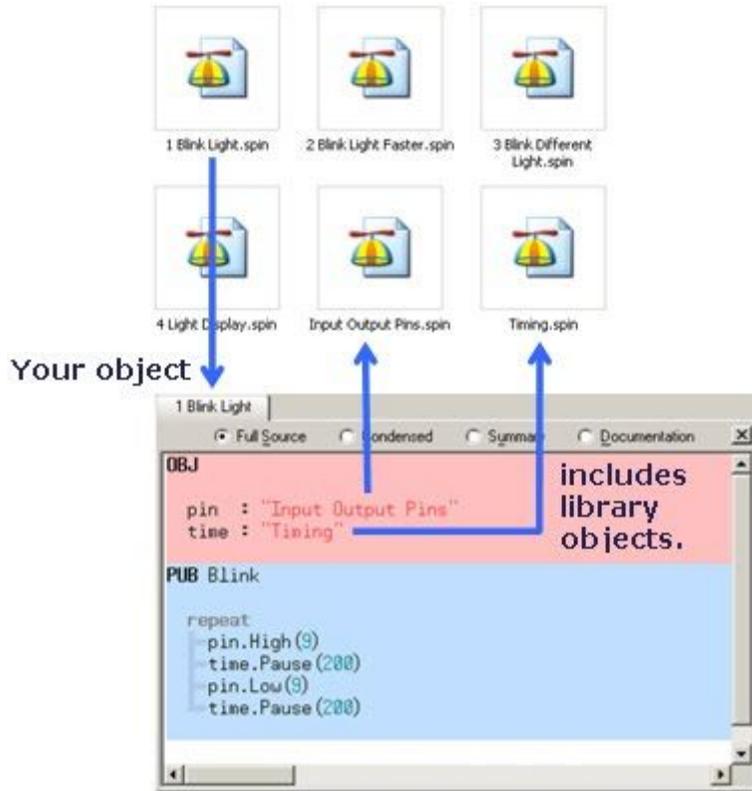
IMPORTANT: Your library objects have to either be in the same folder with your program-object, or in the Propeller Library Folder (typically C:\Program Files\Parallax Inc\Propeller Tool v1.3\Library).

Library objects have built-in documentation that shows you how to use their methods like commands in your program. Let's take a look at the documentation in the Input Output Pins object.

- ✓ Click the Input Output Pins object in the Propeller Tool's top-left Object View windowpane. The file will automatically open in another tab. (If the panes on the left are not visible, click File and select Show Explorer.)
- ✓ Click the Documentation radio button to view the object's documentation.
- ✓ Scroll down and check out the list of methods the object has — they all begin with "PUB." This list will be under the heading Object "Input Output Pins" Interface. Note that it has High and Low methods. (Your program used these.)
- ✓ Scroll down further, and you'll find the High and Low method documentation, which explains how to use it in your code.
- ✓ Repeat this process with the Timing object. Open it and look up the Pause method.
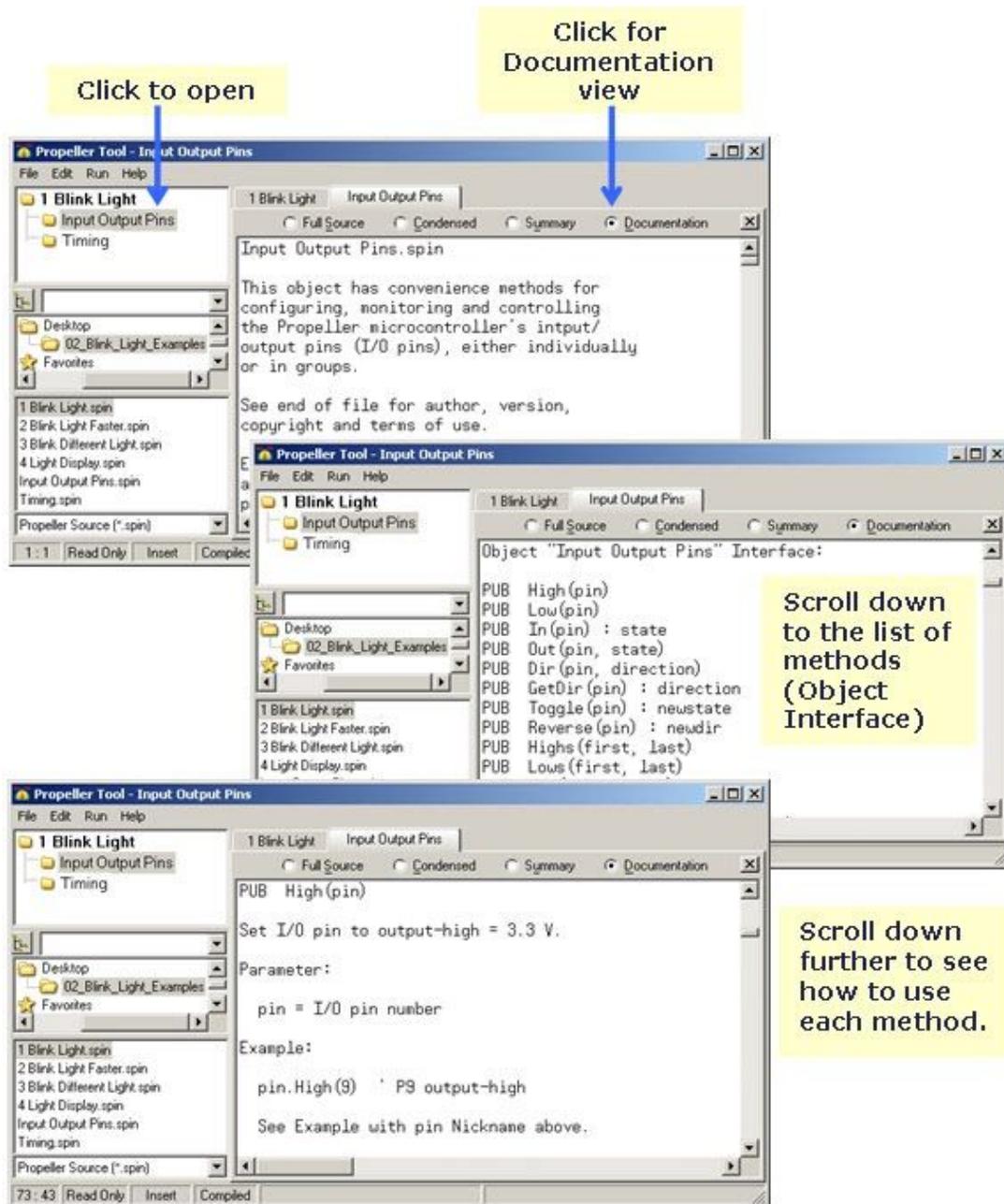
TIP: You can click, hold and drag a tab off the editor so that you can view it in a separate window while you write your code.

Inside the Code

Your program has to tell the Propeller Tool what objects it's going to use. All it takes is a block of code labeled `OBJ`. Inside this block, give each object a nickname, followed by a colon, and then the actual filename in quotes. Then, in your program, you will use the object by its nickname.

Object Block

```
OBJ

   pin  : "Input Output Pins"
   time : "Timing"
```

Object   Object
Nicknames   Filenames
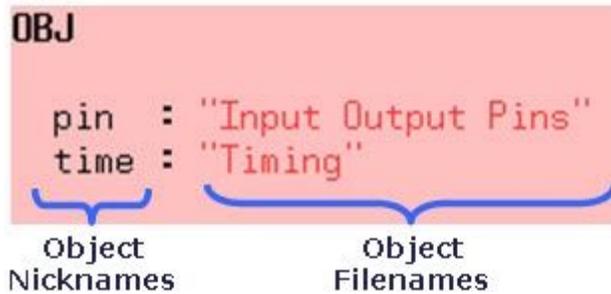
In your code, you can use an object's method like a command by using its nickname, then a dot, then the method name. This is a kind of "method call" and will be referred to as such. In this example, the nickname for the Input Output Pins object is "pin." So, to call its High method and pass its pin parameter the value 9, you would use pin.High(9).

```
PUB Blink

   repeat ···················· Repeats code that's below and indented.
      pin.High(9) ·············· Turn on LED circuit connected to P9.
      time.Pause(200) ········· Wait for 0.2 seconds.
      pin.Low(9) ·············· Turn off LED circuit.
      time.Pause(200) ········· Wait another 0.2 seconds.
```

These "method calls" are below the
repeat command and indented further
than it is. So they so they get
executed over and over again.

This code example also has a Spin language command: repeat. The repeat command can make the Propeller chip execute one or more lines of code containing commands, expressions and/or method calls over and over again. You have to put the lines of code directly below the repeat command, and indented further than it is. The next line of code that's at the same level of indentation as the repeat command will not be part of the loop.

The repeat command has lots of variations for controlling the number of times the loop is repeating, keeping track of an index value, and conditions for repeating. We'll try them all out later. For now, let's focus on experimenting with parameters in calls to the Timing object's Pause method.

Blink the Light Faster

Before making any changes to your code, it's best to save your program with a new name.

- ✓ Click File and select Save As…
- ✓ Type "2a Blink Light Faster" into the Filename field and click Save.
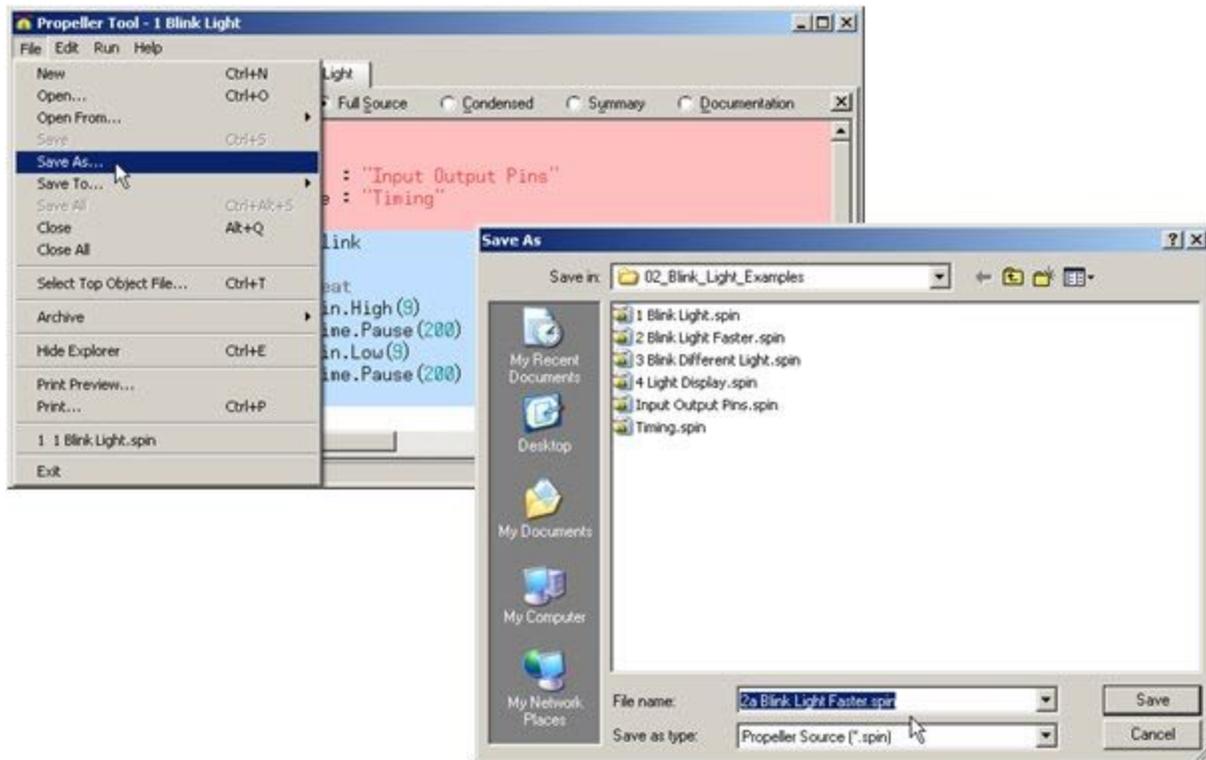
The Timing object's `Pause` method has a duration parameter — a value that says how long to pause for. All you have to do to make the light blink faster is pass a smaller value to this duration parameter. Let's try 100.

- ✓ Change the parameter value you're passing to the Timing object's Pause method from 200 to 100 in both method calls, as shown in the picture.
- ✓ Run the modified program, and then Press the F11 key (a shortcut to Run -> Compile).
- ✓ Verify that the light blinks twice as fast.

Change from 200 to 100

```
PUB Blink

repeat
   pin.High(9)
   time.Pause(200)
   pin.Low(9)
   time.Pause(200)
```

Change from 200 to 100

✓ Save your work, then save your file under another name. To make it easier to find later, start the filename with 2b.

✓ Try passing the Timing object's `Pause` method different duration values. How fast can you make the light blink? What happens if you use different values for each method call?

Individual vs Group I/O Operations

The Input Output Pins.spin object can also control multiple I/O pins and groups of I/O pins.

Modify the Circuit

● Add seven more jumper wires to build the circuit shown here:
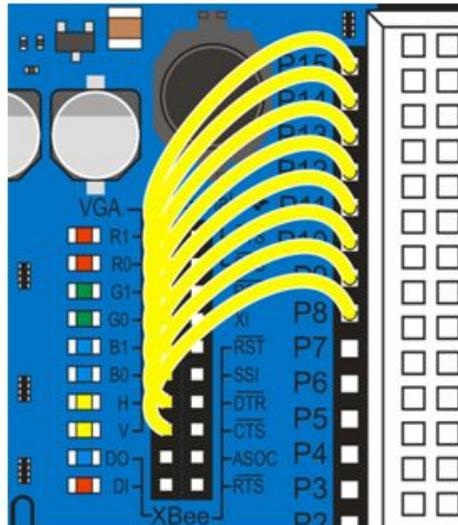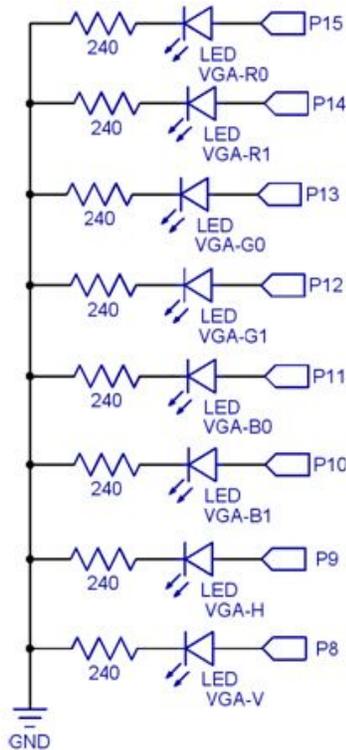
Control a Different Light

The example program "3 Blink Different Light.spin" makes the blue light next to the B1 label blink.

✓ Open 3 Blink Different Light.spin.
✓ Load the program into the Propeller chip with F11.
✓ Verify that the blue light next to the B1 label blinks.

How to Select a Light

Just as the Timing object's `Pause` method expects a number of ms to pause, the Input Output Pins object's `High` and `Low` methods expect an I/O pin number. This number determines which I/O pin sends a high (3.3 V) signal to turn the light on, or a low (0 V) signal to turn it off. This example program passes the value 10 to `pin.High` and `pin.Low`. The previous example program used the value 9. So, instead of sending high and low signals to P9, this program sends them to P10. You used a jumper wire to connect P10 to the B1 socket next to the blue LED circuit.

✓ Save your program under a different name, that starts with 3a.
✓ Try different values (from 8 to 15) in the `pin.High` and `pin.Low` method calls.

Changed from 9 to 10.

```
PUB Blink

    repeat
        pin.High(10)
        time.Pause(100)
        pin.Low(10)
        time.Pause(100)
```

Change from 9 to 10.
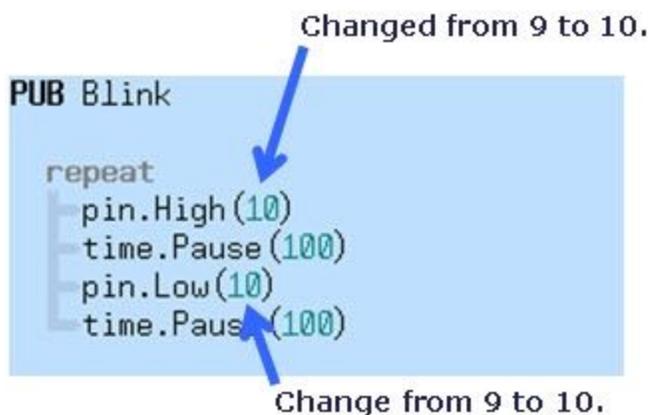
Group I/O Operations

The Input Output Pins object also has methods for controlling, monitoring, and configuring continuous groups of I/O pins. For example, it has both a `High` method, and a `Highs` method. Likewise, there are also `Low` and `Lows` methods. The fact that it's plural (ends in s) is your clue that you can use it to set a group of I/O pins high/low. One of Input Output Pins' more useful methods for group I/O control is `Outs`. The `Outs` method allows you to use a binary value (a number comprised of only ones and zeros) to control the high/low state of each I/O pin in a group. Let's try it.

✓ Examine the Input Output Pins object in Documentation view again.

✓ This time, look up the `Outs` method.
✓ Next, open "4 Light Display.spin" with the Propeller Tool software.
✓ Load the program into the Propeller chip with F11, and verify that it makes the lights blink in an alternate on/off pattern.

How the Spin Code Works

The Input Output Pins object's `Outs` method expects three numbers: the first pin number in the group of I/O pins, the last pin number, and a binary number that describes the high-low pattern. In Spin, you indicate a binary number with a percent sign in front, like this: `%01010101`. Here's what that does to the high/low states of the group of I/O pins starting with P15 and ending with P8.

```
PUB Blink

    repeat
      pin.Outs(15,8,%10101010)
      time.Pause(400)
      pin.Outs(15,8,%01010101)
      time.Pause(400)
```

First (leftmost) binary digit sets state of P15

P8 high
P9 low
P10 high
P14 high
P15 low

Last (rightmost) binary digit sets state of P8
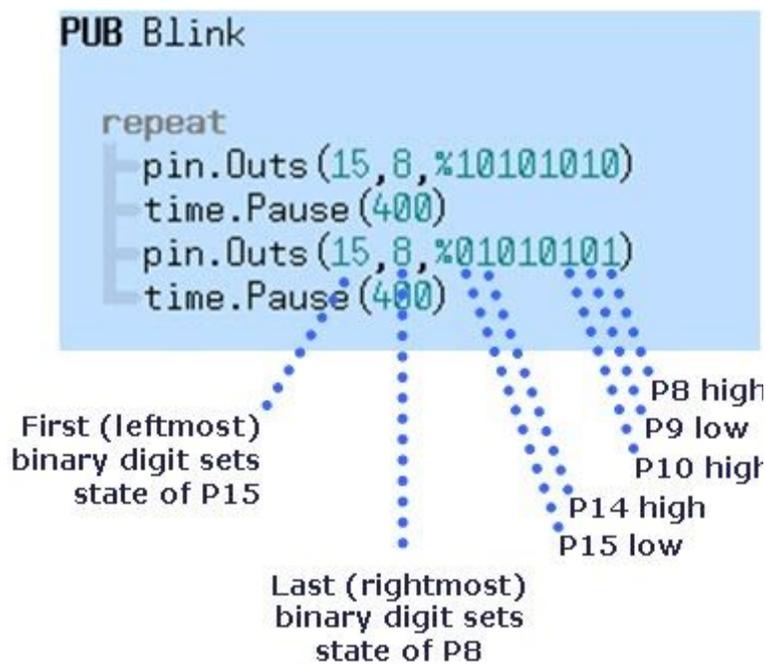
Your Turn – A Different Pattern

Let's try turning 4 lights that are next to each other on, while the other four are off. Your binary values would have to be `%11110000` and `%00001111`.
✓ Try it!

Lesson 3: Spin Programming

This next set of activities will show you how to write programs that perform simple, common programming tasks. Examples include displaying messages and values, counting, calculations, decision making, and using objects instead of writing lots of code from scratch.

All example programs will display results with the Parallax Serial Terminal software that comes bundled with the Propeller Tool.
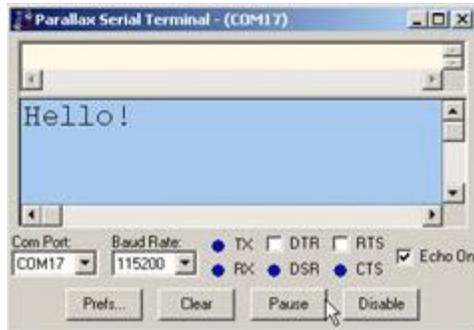

✓ Follow the sections below.

Hello from your Propeller

Let's start with a few simple example programs that make your Propeller Board of Education's microcontroller send some simple messages to your PC. On the PC side, we'll use the Parallax Serial Terminal software that accompanies the Propeller Tool to receive and display those messages.

Open and Run First Example

✓ Open the PropBOE_Spin_Intro_20120130 folder.
✓ Find and double-click 1 Hello Propeller.spin. It should automatically open into your Propeller Tool software.
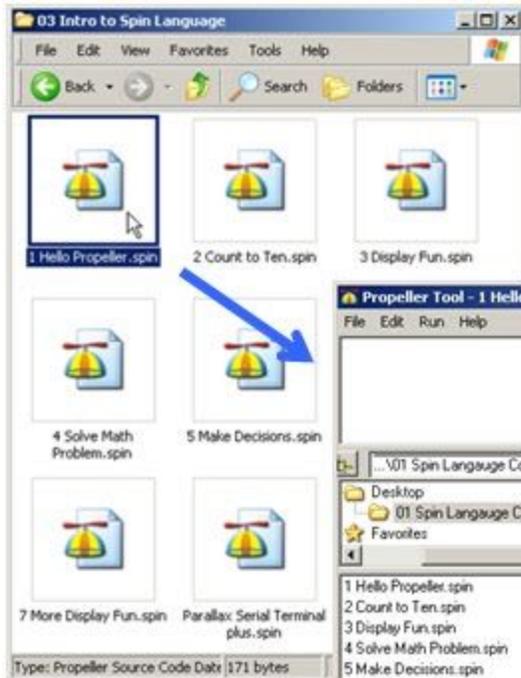
Double Click 1 Hello Propeller to open it with the Propeller Tool software.
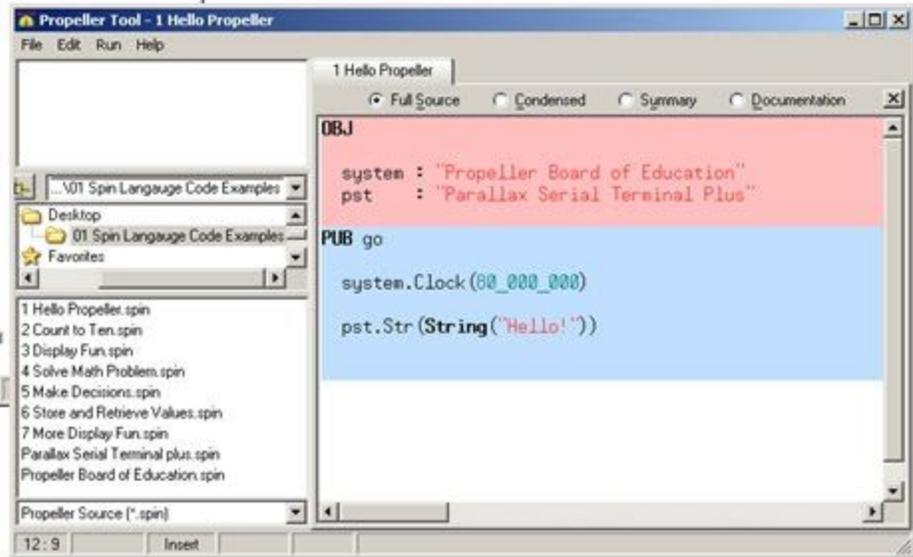
✓ If you closed your Parallax Serial Terminal, you may need to re open it and set the Com Port again. See the Run the Software and Configure the Software sections in Lesson 1.
✓ Click Run, then point at Compile Current, and then click Load EEPROM. (F11 is the shortcut for loading your program into EEPROM.)
✓ As soon as you see the Programming EEPROM" message, click the Enable button on the Parallax Serial Terminal's bottom right corner.

TIP: As soon as you see the "Programming EEPROM" message, click the Parallax Serial Terminal's Enable button.
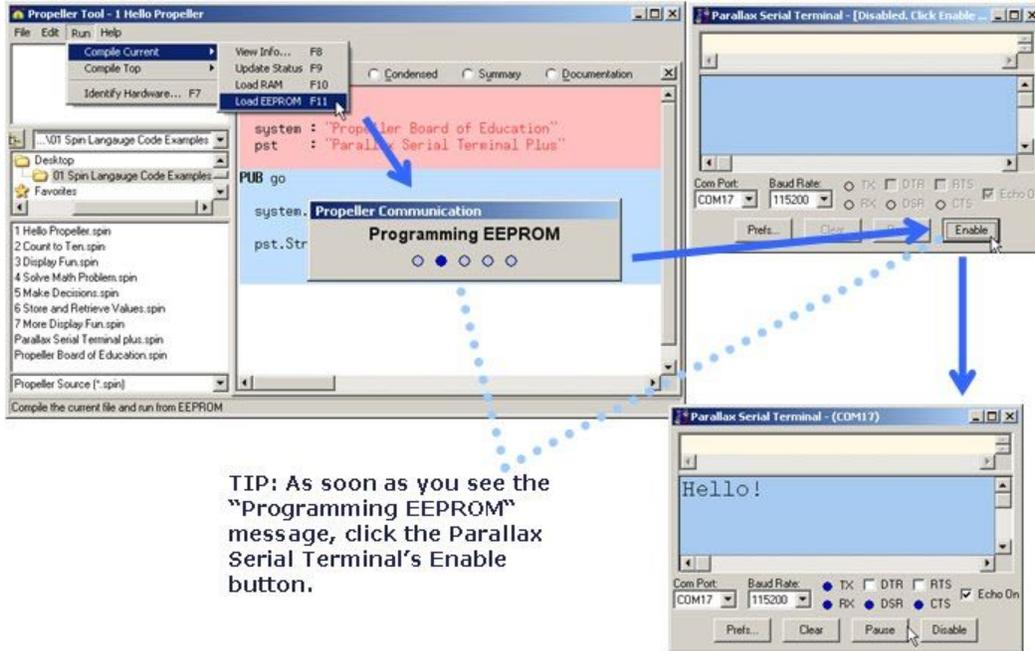
More About Library Objects

The "Hello!" example uses a new pair of library objects, named "Propeller Board of Education" and "Parallax Serial Terminal Plus."

Your Object

Library objects
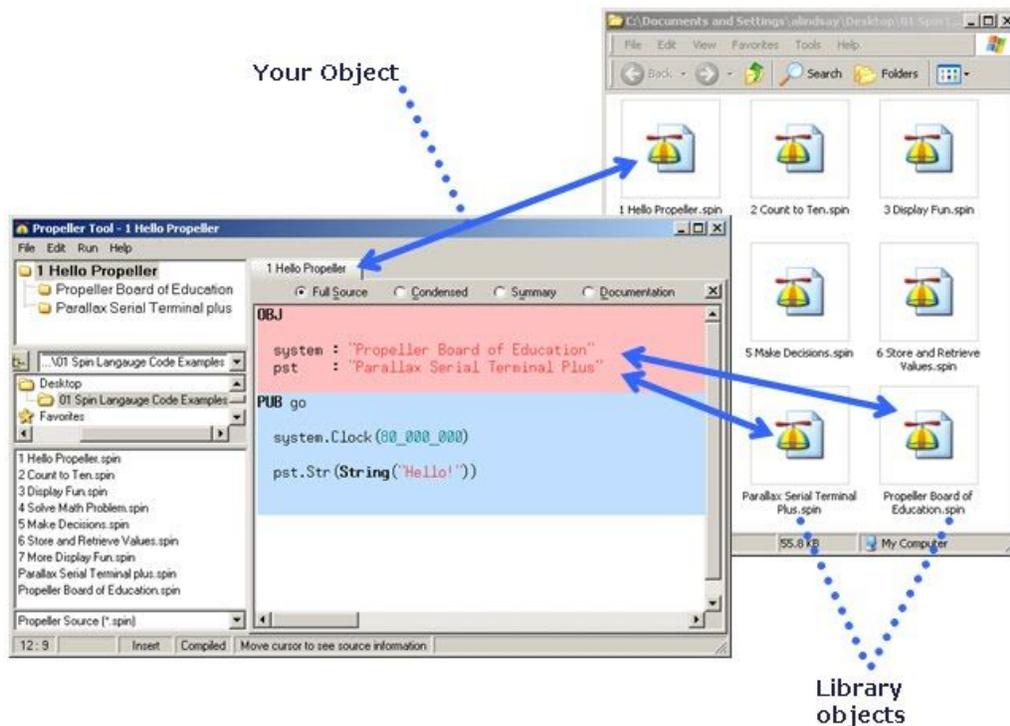
Code Block and Object and Method Review

Here is a second look at declaring objects and using their methods in your program. This program declares the Propeller Board of Education object and gives it the nickname system. Then, it calls the Propeller Board of Education's Clock method with `system.Clock(80_000_000)`. This program also gives the Parallax Serial Terminal Plus object the nickname `pst` and then calls its `Str` method, which sends the "Hello" message to your computer. `Str` is one of the many methods Parallax Serial Terminal Plus offers for communication with your computer.

- Examine the the steps in the figure carefully; you will be using them to make your own programs.
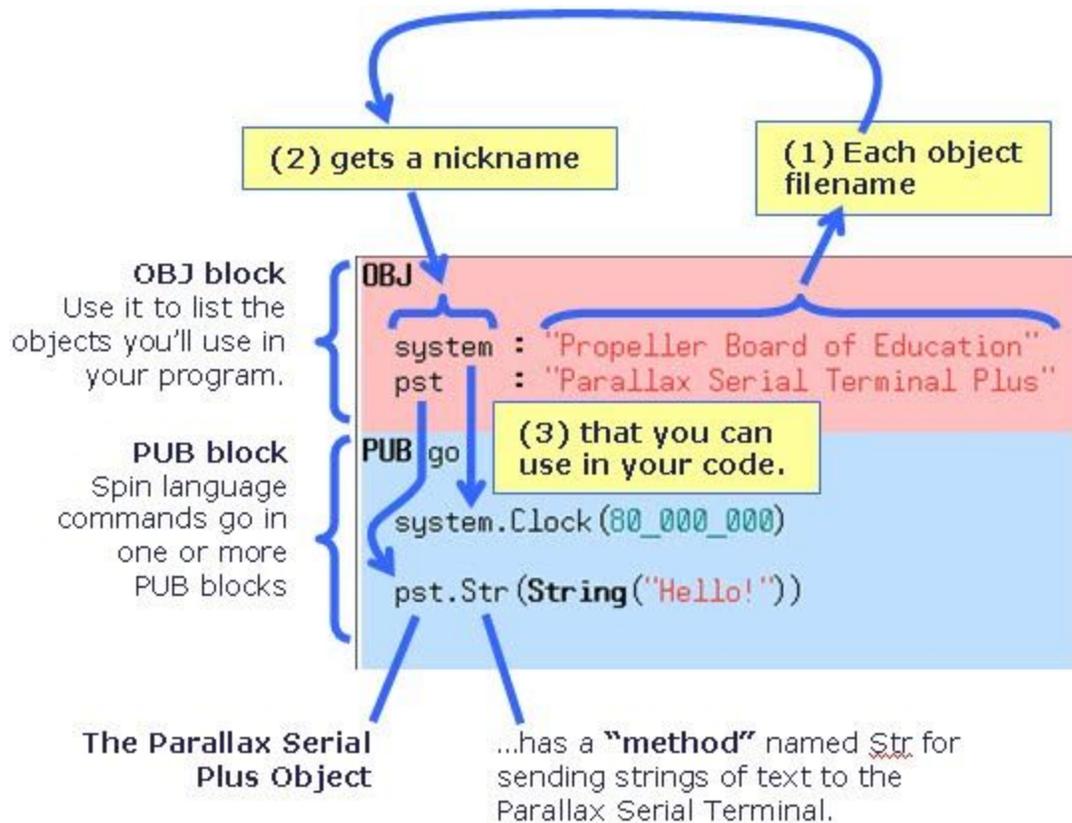
Did you know?

You can use `system.Clock` to set the Propeller chip's clock to these frequencies: 80 MHz, 40 MHz, 20 MHz, 10 MHz, and 5 MHz. If you don't set the Propeller chip's system clock, it will default to a 12 MHz internal clock that's not very accurate. It's great for some applications, but not precise enough for serial communication.

The "Parallax Serial Terminal Plus" object uses another one of the Propeller chip's eight core processors (called cogs) to communicate with your PC. The `pst.Str` method call passed the

"Hello!" message to the Parallax Serial Terminal object, and code in that object passed it to a different cog that transmitted each character to your computer.

New Tip: View Object Documentation in another Window

It really helps to be able to look at an object's methods in a different window while you are writing your code. Here's how to do that:

- ✓ Click Parallax Serial Terminal Plus in the upper-left Object View pane to open the Parallax Serial Terminal Plus object in a new tab.
- ✓ Click the Documentation radio button to change to Documentation view.
- ✓ Click and hold the Parallax Serial Terminal Plus tab, and drag it off the editor to open it into a new window.
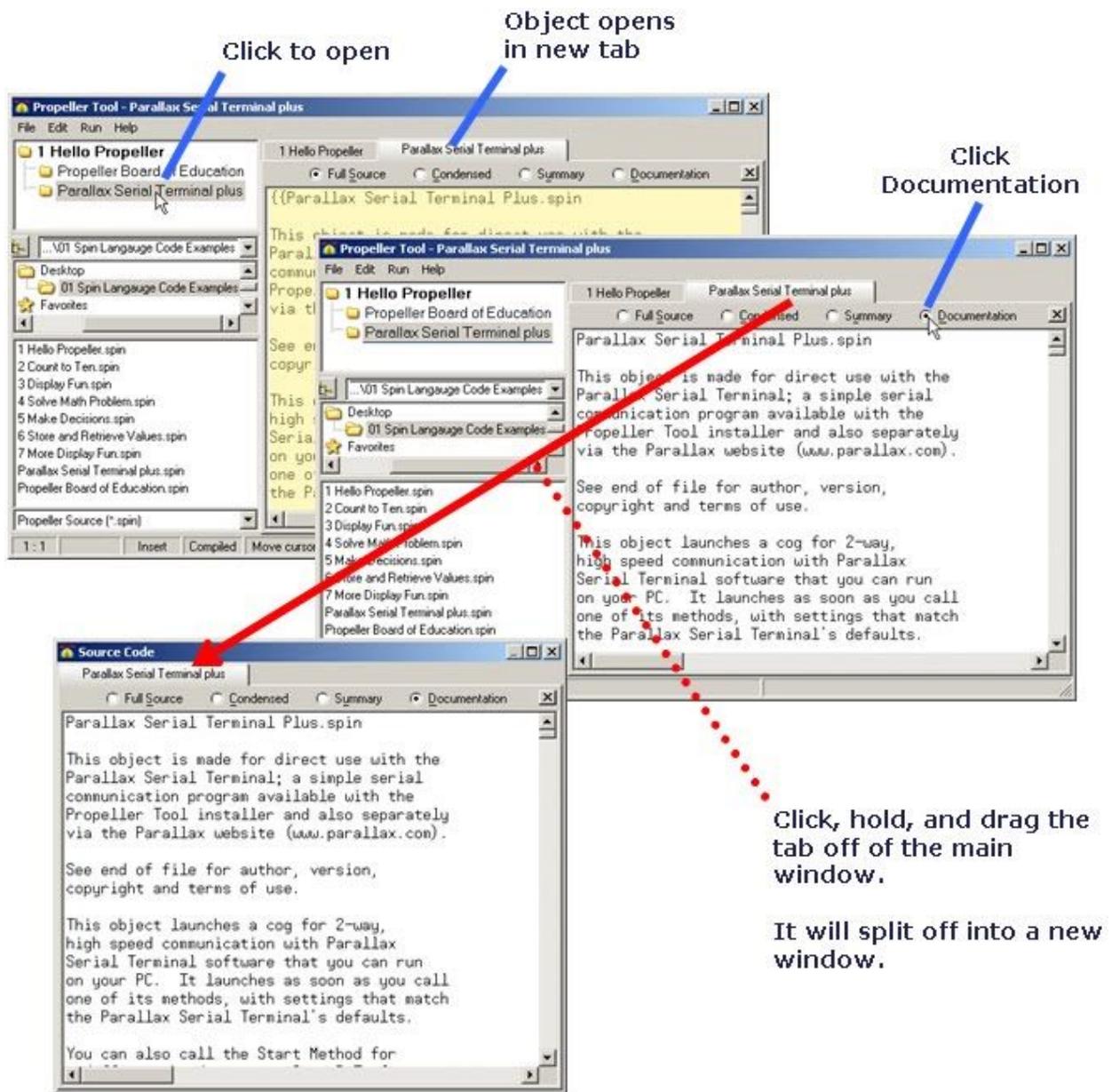
Now, you can check the Parallax Serial Terminal Plus Object Interface and method documentation in a separate window while you write your code. It's also a good idea to arrange the windows so that you can view them both on your screen at the same time. This will make it a lot easier to find methods, read about them, and add them to your code.

- ✓ Arrange the windows so that you can see both your example program and the Parallax Serial Terminal Plus object (which is in Documentation view).
- ✓ Examine the list of methods in the Object "Parallax Serial Terminal Plus" Interface. Do some of the names look familiar? Since `Str` is short for string and the method sends a string, what do you think `NewLine` does? How about `Dec`?
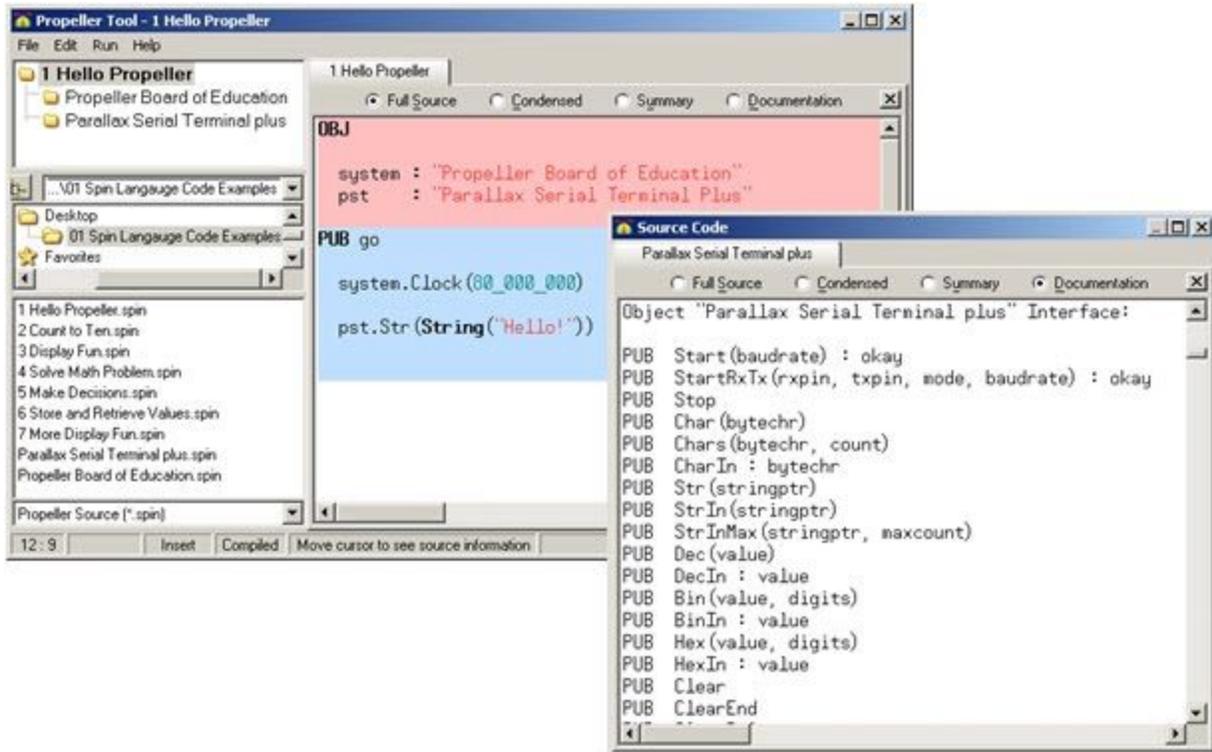
Answering the questions about the `Str, Newline, and Dec` methods is easy, just scroll down and read the documentation for each one. Each method is fully documented below the Method Interface, which is just a list of their names.

✓ Scroll down and find the `Str, NewLine, and Dec` method documentation.
✓ Read them, and then let's try using the `NewLine` method to display a string on a second line. We'll also use the `Dec` method in the next activity.

Scroll down to find Str method Documentation.

```
Source Code                                    _ □ X
 Parallax Serial Terminal plus

   ○ Full Source   ○ Condensed   ○ Summary   ● Documentation   X
_____
PUB   Str(stringptr)

Send zero terminated string.
Parameter:
  stringptr - pointer to zero terminated
              string to send.

Examples:
  ''Send string with String operator.
  pst.Str(String("Hello!"))

  ''Send string from DAT block
  '...code omitted
  PUB Go
    pst.Str(@myDatString)
  DAT
    myDatString byte "abcdefg", 0
  '
  '           Zero terminator
```

Scroll down further to find Dec method documentation.
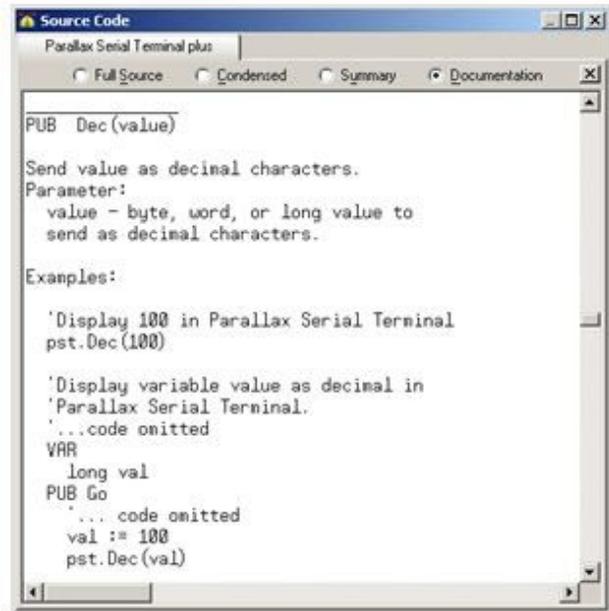
```
Source Code                                    _ □ X
 Parallax Serial Terminal plus

   ○ Full Source   ○ Condensed   ○ Summary   ● Documentation   X
_____
PUB   Dec(value)

Send value as decimal characters.
Parameter:
  value - byte, word, or long value to
  send as decimal characters.

Examples:

  'Display 100 in Parallax Serial Terminal
  pst.Dec(100)

  'Display variable value as decimal in
  'Parallax Serial Terminal.
  '...code omitted
  VAR
    long val
  PUB Go
    '... code omitted
    val := 100
    pst.Dec(val)
```

...and even further to find NewLine method documentation (not shown).

What's a "Zero Terminated String"?

The expression `String("Hello!")` stores these characters in program memory, with a zero at the end.

- Click the "1 Hello Propeller" tab in the Propeller Tool.
- Click Run -> Compile Current -> View Info... F8.
- The Object Info window that appears has a Show Hex button. It should resemble this one. Find the Hello characters.

Hello character codes with a zero at the end

Hello characters
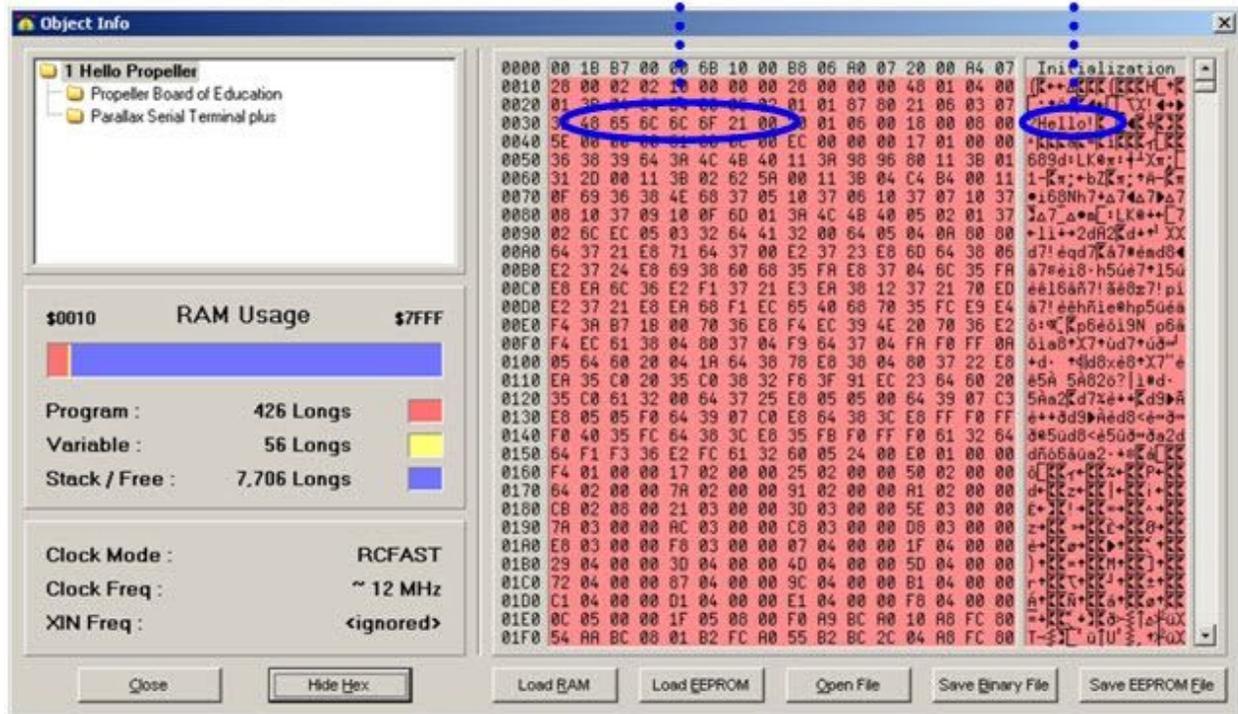
Display a Value

We've already checked the Parallax Serial Terminal Plus object's NewLine and Dec methods, so let's try it out.

- ✓ Save your example program under a new name, like "1a Hello Propeller.spin"
- ✓ Double check the documentation for NewLine and Dec.
- ✓ Change the "Hello!" text to "The number is:"
- ✓ Add two new lines after the `pst.Str` call:

```
pst.NewLine
pst.Dec(42)
```

- ✓ Run the modified program

Tip: use the F11 key, and remember to click the Parallax Serial Terminal's Enable button right away.

Add:
pst.NewLine
pst.Dec(42)

Modify the
text string

F11 to load
the
Program

Remember to click
the Enable button
right away!
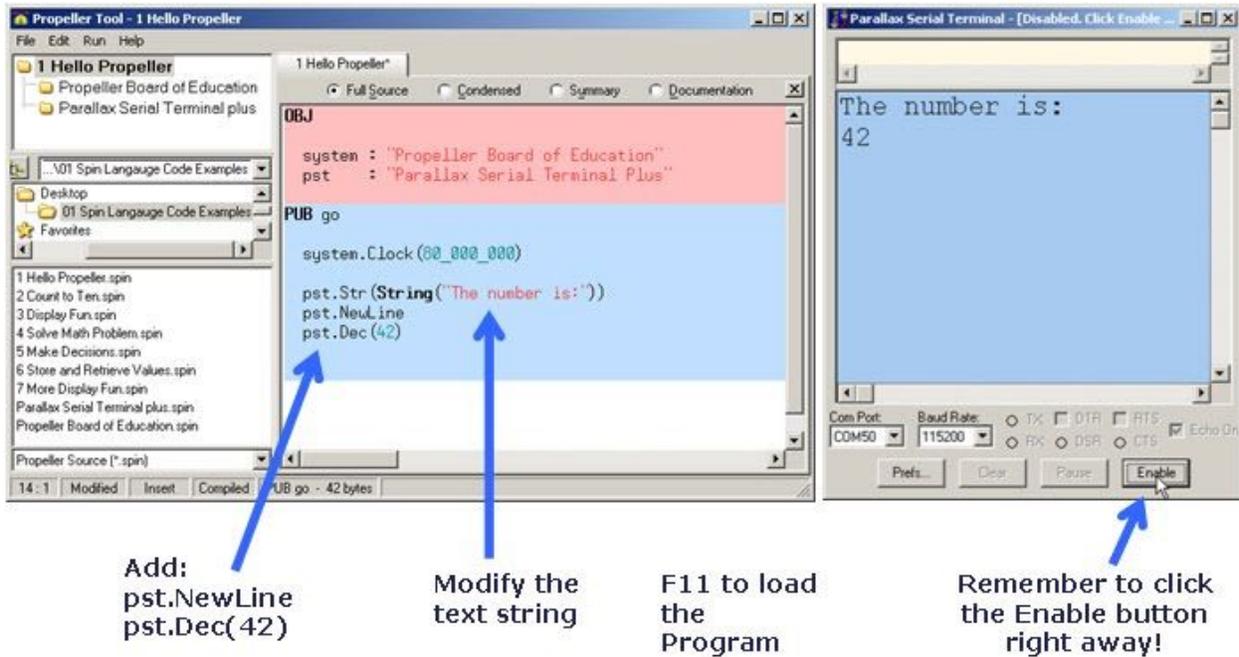
Count to Ten

Let's try a program that counts to ten.

✓ Go back to the folder you unzipped and double-click "2 Count to Ten.spin" to open it.
✓ While in the Propeller tool, press F11 to load the program into the Propeller chip.
✓ Make sure to click the Parallax Serial Terminal's Enable button right away.
✓ Remember, you don't have to wait until the program is finished loading.

**Press F11 to Load the program into the Propeller Chip**
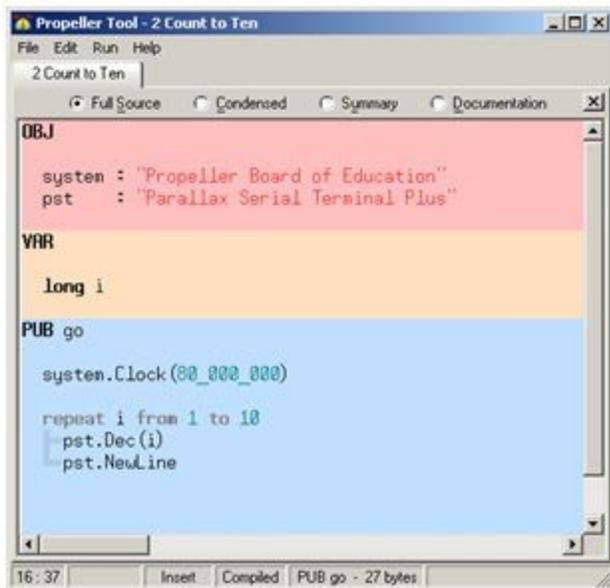
Remember to click the Enable button, even while the program is still downloading
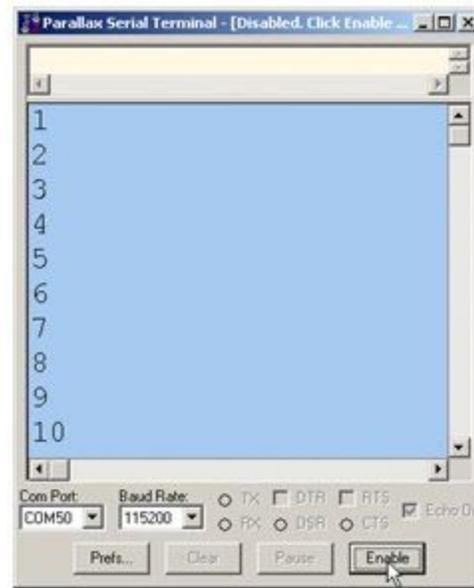
Now, what if you want to make the program count from 8 to 15 instead? In case you're wondering why we would want to count from 8 to 15, try the Project at the bottom of this section. The first step is to save the file to a new name before making any changes.

    ✓ In the Propeller Tool software, click File and select Save as

   ✓ For a new filename, use "2a Count from Eight to Fifteen.spin".
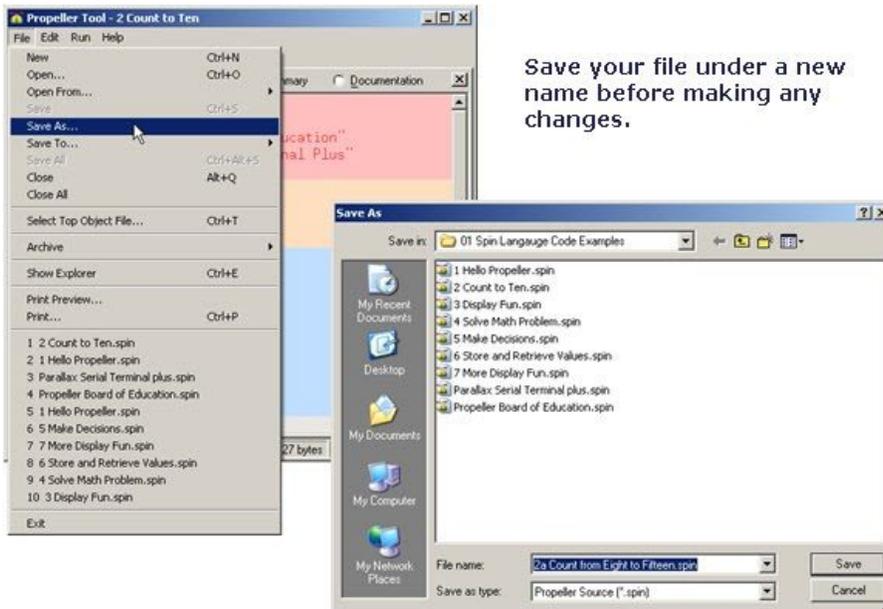
Save your file under a new name before making any changes.

Now, modify the program and run it to test the effect.

    ✓ **Change** `repeat i from 1 to 10` **to** `repeat i from 8 to 15.`

✓ Run the program (load it into the Propeller chip by pressing F11 and then click the
Parallax Serial Terminal's Enable button).

✓ In the Propeller Tool, click File -> Save (or CTRL+S) to save your work.
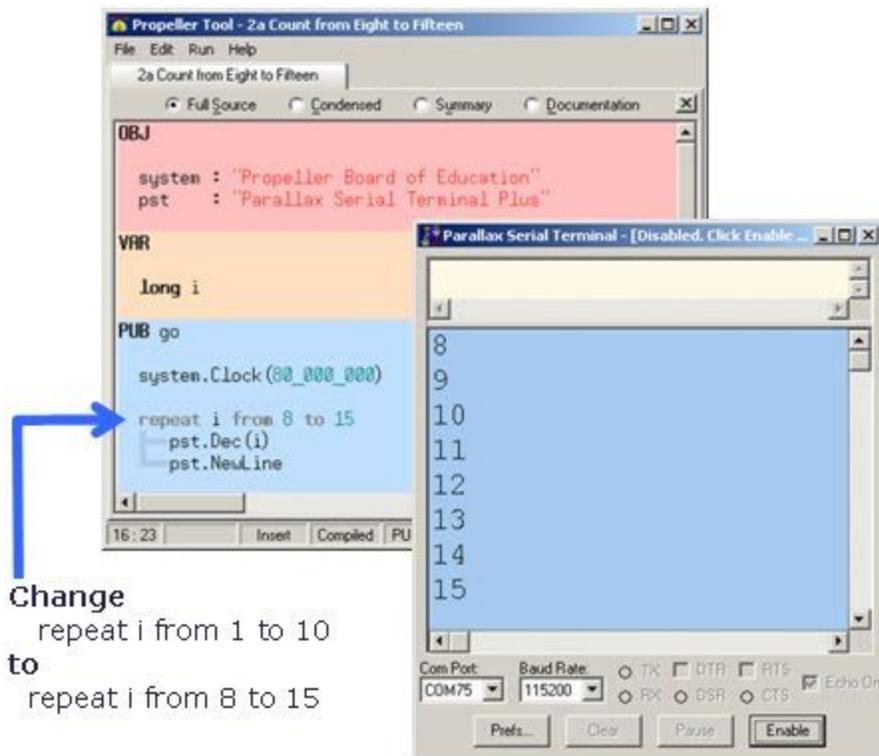
You can also modify that line of code to make it count in steps, let's try steps of 2.

✓ Save the program under a new name again. This example will use "2b Count in Steps of
2.spin".

✓ Change `repeat i from 8 to 15` to `repeat i from 8 to 15 step 2`.

✓ Run the program and observe the effect.

Your Turn: More variations on counting

If you don't need any kind of index in your loop, you can just use `repeat` and a number. For
example, to make code that's in the `repeat` loop execute eight times, use:

```
repeat 8
```

For example, if you just want to print eight copies of a "Hello!" message, you could use:

```
repeat 8
  pst.Str(string("Hello!"))
  pst.NewLine
```

✓ Try it!

Project: Make a Bar Graph

Use what you have learned about indexed counting to go back to and make a bar graph using the circuit from the Individual vs. Group I/O Operations section of Lesson 2.

- ✓ If you have already removed the wires that controlled the Lesson 2 LEDs, go back to Individual vs. Group I/O Operations and rebuild the circuit.
- ✓ Start with Blink Different Light.spin. It should be in the 02_Blink_Light_Examples folder.
- ✓ Save it under a new name, then add a `VAR` block and declare an index variable, like this:

```
VAR long i
```

- ✓ Your code in the `Blink` method should look like this:

```
PUB Blink
  repeat i from 8 to 15
    pin.High(i)
    time.Pause(100)
  repeat i from 15 to 8
    pin.Low(i)
    time.Pause(100)
```

- ● Run the example and observe the effect. Note also that you can count downward with

```
repeat i from 15 to 8.
```

Simple Math Problems

This activity uses Propeller to do math problems, by entering numbers and displaying answers with the Parallax Serial Terminal.

The example program "03 Solve Math Problems.spin" prompts you to "Enter value a", and waits for you to type a value into the Parallax Serial Terminal's Transmit windowpane and press the Enter key. It then asks you to "Enter value b", and you'll have to type another value and press Enter. It then adds the two values and displays the sum.

- ✓ Double-click "3 Solve Math Problems.spin" to open it into the Propeller Tool.
- ✓ If you want to hide the Explorer pane so that your Propeller Tool matches the picture, click File and Select Hide Explorer.
- ✓ Press the F11 key to load the program into the Propeller chip.
- ✓ Click the Parallax Serial Terminal's Transmit windowpane.
- ✓ Follow the prompts, typing values and pressing enter after each one.
- ✓ Verify that it displays the sum of the two values you entered.

Click the Transmit windowpane, then type each value and press Enter.
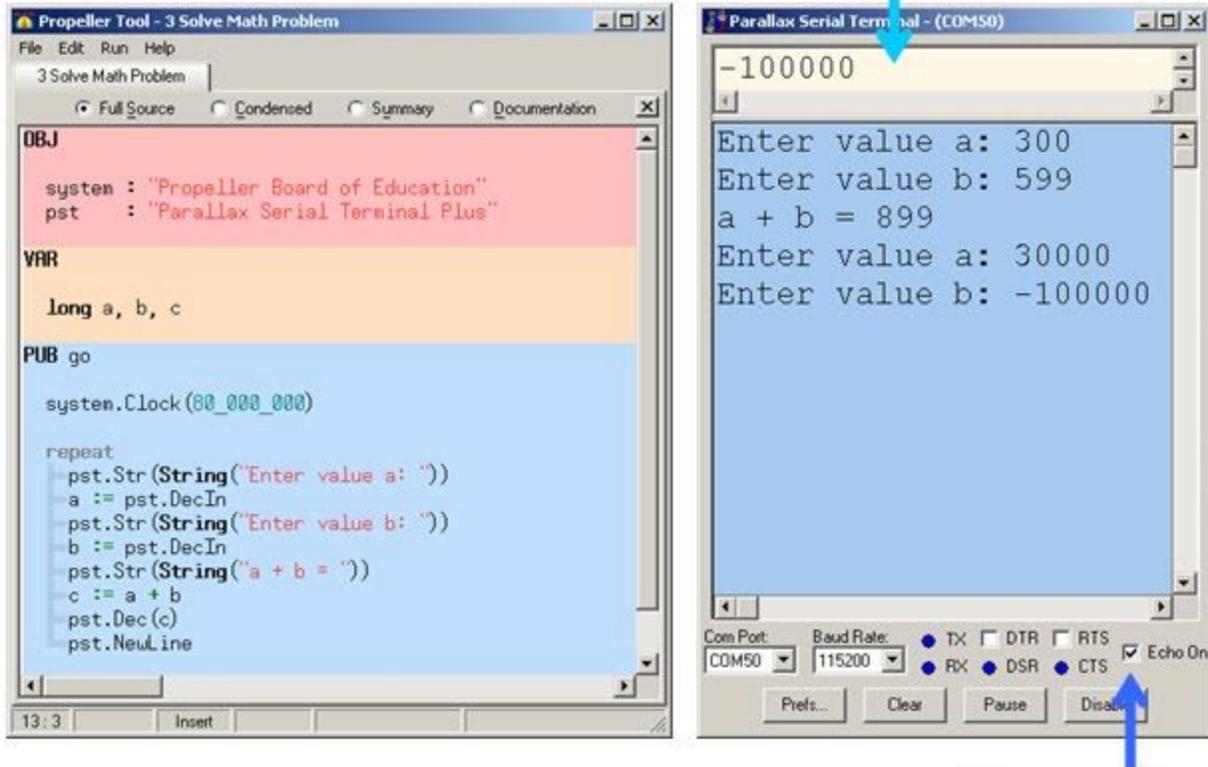
Make sure Echo On is checked.

How it Works

The `pst.DecIn` method call waits for you to type a value into the Parallax Serial Terminal's Transmit windowpane. When you press Enter, it figures out the decimal equivalent of the characters you typed, and returns that value. So, the expression `a := pst.DecIn` waits for you to type that value and press Enter. As soon as you do, it copies the value you typed into a variable named a. The command `b := pst.DecIn` copies a second value into a variable named b. Then, `c := b + a` adds the two values together and stores the result in the variable named c. After that, the program displays the result with `pst.Dec(c)`.

Your Turn: Try Different Operators

There are lots of math operators you can use in place of the + in c := a + b. For example, you could use the subtract, multiply, divide, or remainder operators (-, *, /, //).

   ✓ Save 3 Solve Math Problems.spin as 3a Test Subtraction.spin.

- ✓ Change `c := a + b` to `c := a - b`.
- ✓ Change `pst.Str(String("a + b = "))` to `pst.Str(String("a - b ="))`.
- ✓ Run the modified program and verify that it now subtracts `b` from `a`.
- ✓ Repeat for the multiply, divide, and remainder operators.

Learn More about Operators

The Spin language has more operators you can use. The Propeller Manual has more info.
- ✓ In the Propeller Tool Software, click Help and select Propeller Manual.
- ✓ Look up Operators and Binary Operators in the index and read about them.
- ✓ Look up Binary Operators in the index, and take a look at the list.

Note: The five operators we just experimented with are examples of *binary operators*. They are called binary operators because they perform an operation on two values.

Challenge!

- ✓ Modify Solve Math Problems.spin so it divides two numbers and then displays both the result and the remainder. Be sure label the remainder value in the display!

Simple Decisions

The "4 Make Decisions.spin" example program prompts you to enter values into the Parallax Serial Terminal's Transmit windowpane. After you press Enter, the program decides whether the number is positive, negative, or zero.
- ✓ Open "4 Make Decisions.spin" and use F11 to load it into the Propeller chip.
- ✓ Try typing various negative and positive values into the Parallax Serial Terminal's Transmit windowpane. Remember to press Enter after each value.
- ✓ Verify that the program correctly identifies negative, positive and zero values.

Click the Transmit windowpane, then type each value and press Enter.
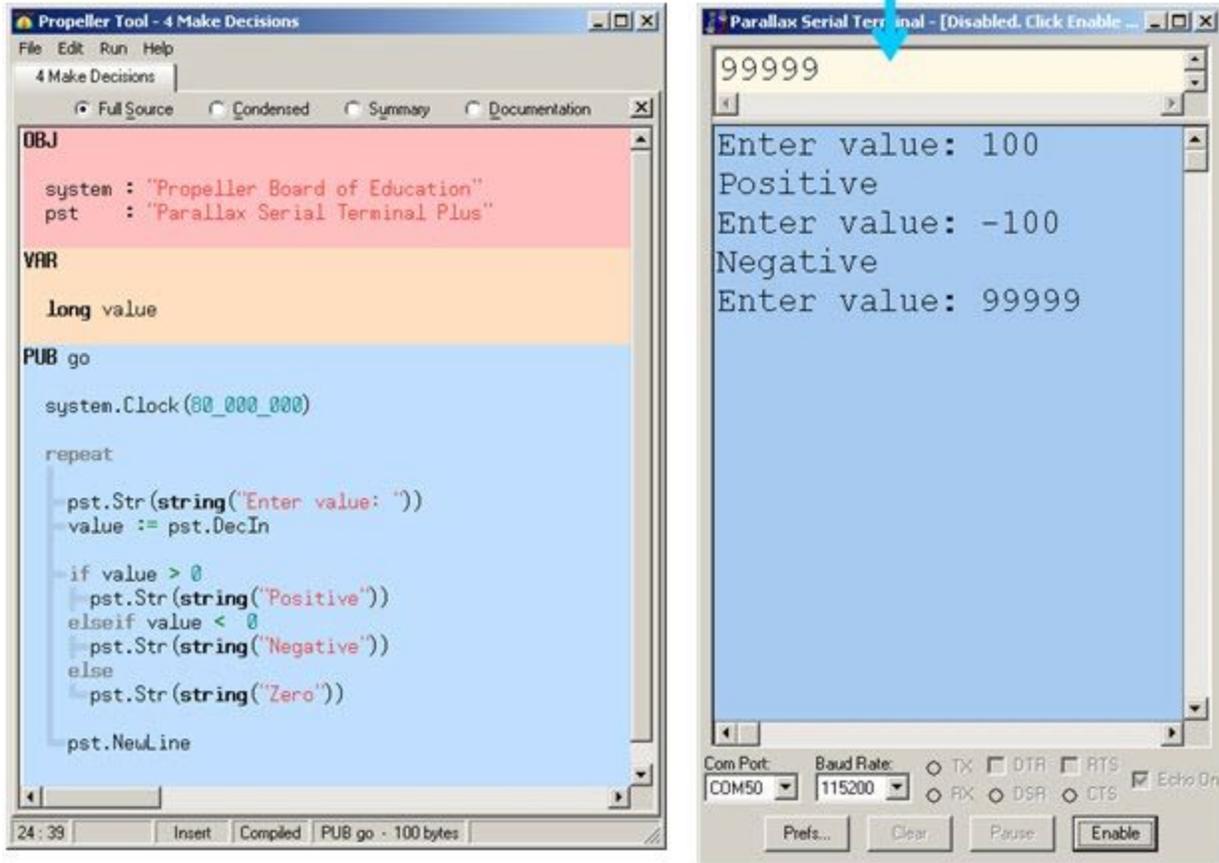
How it Works

The program copies the number you entered to a variable named `value` with `value :=` `pst.DecIn`. Then, it analyzes the value with an `if…elseif…else` block of code. It starts by checking if `value > 0`, in which case, it sends the message "Positive" and skips the rest of the tests. If value is not greater than zero, it moves on to check if `value < 0`. If it is, it prints "Negative" and skips any more decision making. If it's not less than zero either, it moves on to the `else` statement, which covers the rest of the possibilities. In this case, the only other possibility is that the `value` variable is zero.

Your Turn

You could accomplish the same thing with three different IF commands. Try this:

    ✓ Save 4 Make Decisions.spin as 4a Make Decisions.spin.

✓ Replace this code:

```
if value > 0
  pst.Str(string("Positive"))
elseif value < 0
  pst.Str(string("Negative"))
else
  pst.Str(string("Zero"))
    ...with this code
if value > 0
  pst.Str(string("Positive"))
if value < 0
  pst.Str(string("Negative"))
if value == 0
  pst.Str(string("Zero"))
```

✓ Test the modified program.
✓ Can you modify it to test if the value is greater than 100, less than -100 or between the two values?

Learn More about IF

Remember, you can view the Propeller Manual from the Propeller Tool software by clicking Help and selecting Propeller Manual.

✓ Open the Propeller Manual and use the index to look up the IF command.

Store and Retrieve Values

Need to store a list of values? Use an array variable. Try this program out.

✓ Open "5 Store and Retrieve Values.spin" with the Propeller Tool.
✓ Use F11 to load it into the Propeller Chip, and remember to click the Parallax Serial Terminal's Enable button while the program is still loading.
✓ Follow the prompts, and type values of your choosing into the Parallax Serial Terminal's Transmit windowpane.
✓ Verify that it lists the values you entered after `I = 5`.

Click the Transmit windowpane, then follow the prompts. Remember to press Enter after each value.
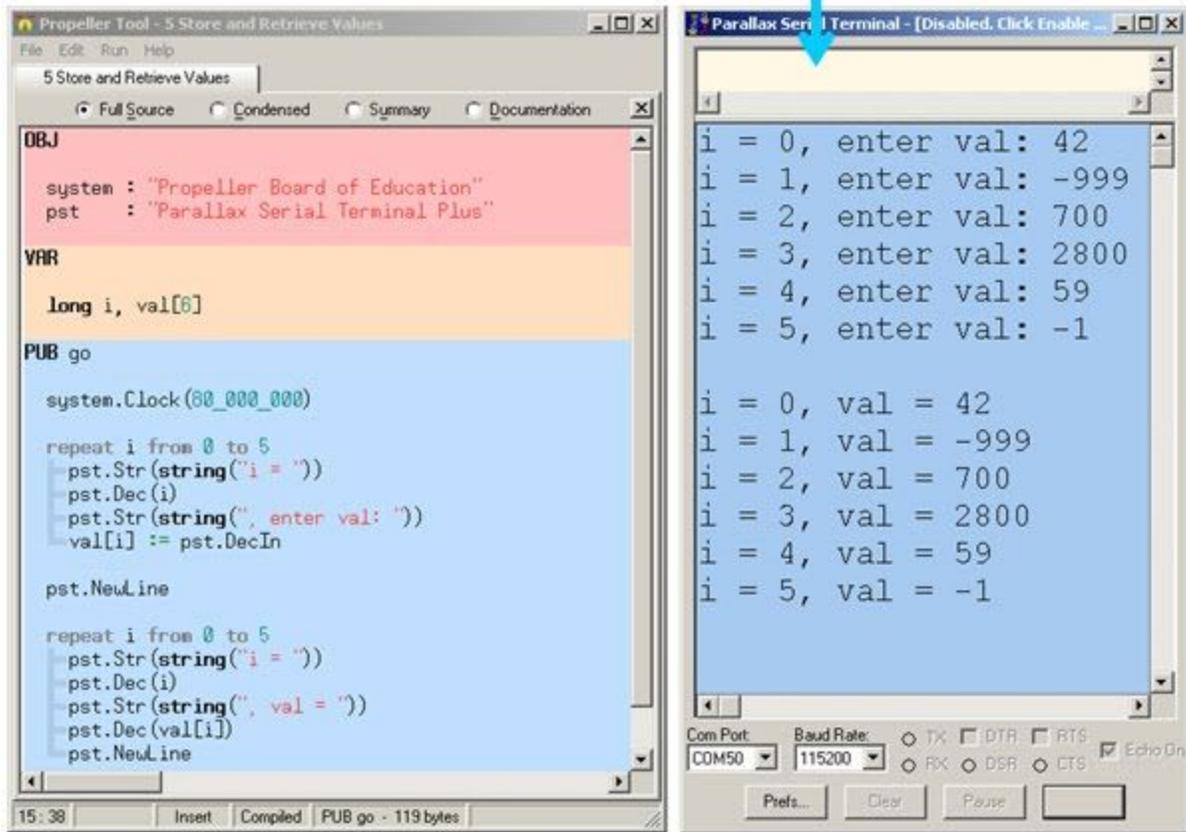
How it Works

In the `VAR` block, the program declares `long i, val[6]`. This declares 7 variables: `i, val[0], val[1], val[2], val[3], val[4]`, and `val[5]`. The program then uses a `repeat` loop with `i` as its index to successively load each variable with a value you type into the Parallax Serial Terminal's Transmit windowpane. After that, a second `repeat` loop uses the same technique to display all the values that are stored.

Your Turn

How about displaying the values in reverse order? Simply swap the 0 and 5 in the last `repeat` loop.

✓ Try it!

Display Fun

"6 Display Fun.spin" displays a line of asterisks, starting at the lower-left and ending at the upper-right.

✓ Open "6 Display Fun.spin", load it into the Propeller chip with F11, and click the Parallax Serial Terminal's Enable button to view the output.
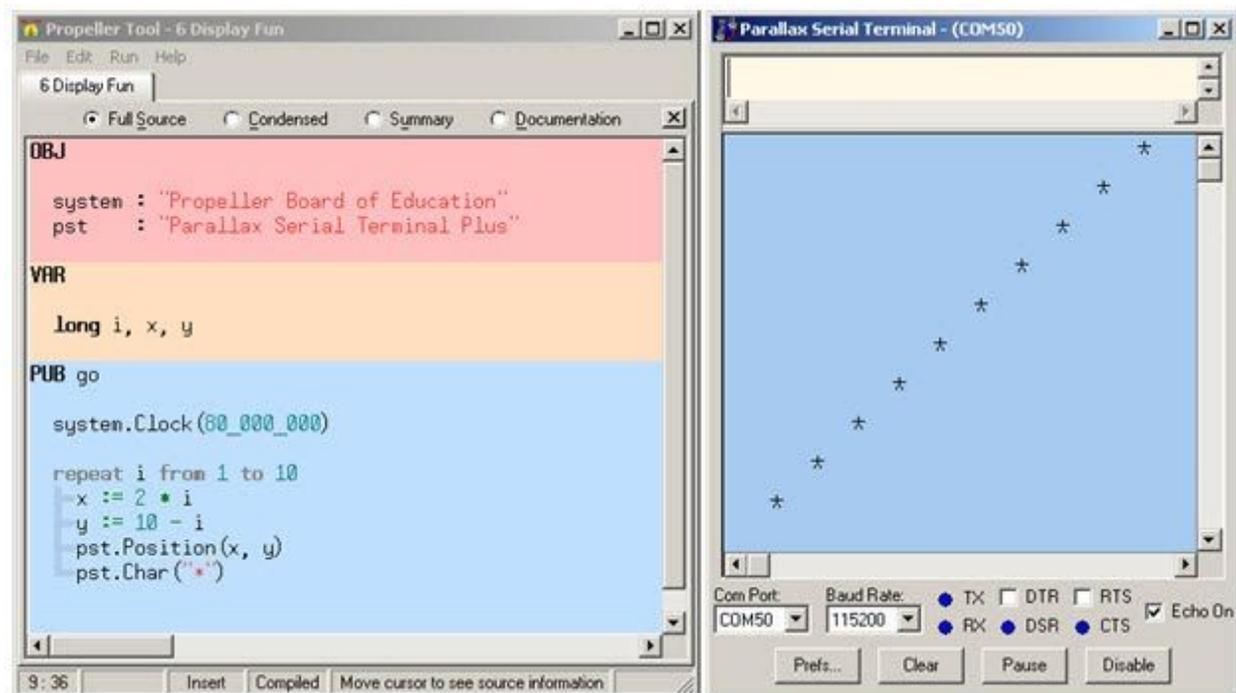
How it Works

This program is combines many of the other things we've learned in this lesson. It uses two more methods from the Parallax Serial Terminal Object: Position and Char, multiply and subtract operators, and indexed counting to display a line of asterisk characters in the Parallax Serial Terminal.

✓ Open Parallax Serial Terminal Plus and view it in documentation mode.
✓ Read up on the Char and Position methods.
✓ Examine the rest of the code, and make a narrative of how it works.

Your Turn

✓ Modify the program so that it draws a large X with asterisks. You're half way there already, but it might take some tinkering.

More Display Fun

You can make large, predefined lists with `DAT` blocks. You will see numerous applications of this in the lessons. This first application is simply storing some character art.

   ✓ Open "7 More Display Fun.spin", load it into the Propeller chip with F11, and click the Parallax Serial Terminal's Enable button to view the output.
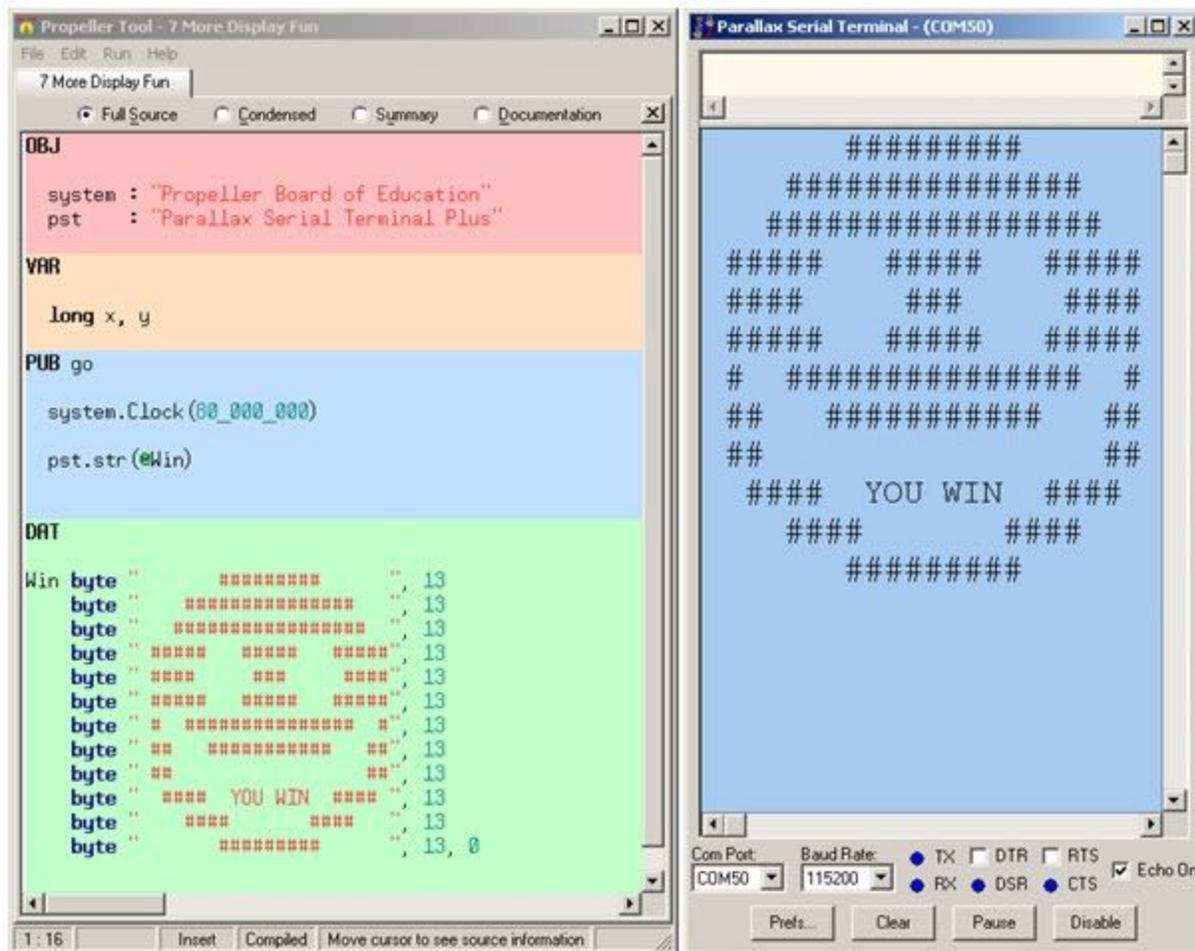
How it Works

Unlike the `String` operator, which stores a list of characters in program memory, the `DAT` block is more like a predefined variable array. The values are loaded into RAM, and can even be modified while the program is running, much like an array.

Remember that the Parallax Serial Terminal Plus object's `Str` method's `stringptr` parameter expects the memory address of a zero-terminated string. Take a look at the last value in the `DAT` block, it's zero. Also, take a look at the `pst.Str` call. It's `pst.Str(@Win)`. The @

operator in `@Win` gives the `Str` method the memory address of the first byte (a space character) at the beginning of the rows of characters. So, the `Str` method does its job of fetching and transmitting characters until it runs into that zero at the end of the list.

Did you Know?

The 13 character code makes the Parallax Serial Terminal's cursor jump to the beginning of the next line.

Your Turn

You can also treat `DAT` blocks like arrays. For example, here is a loop that sends each character in the `DAT` block, one at a time.
- ✓ Save the program as 7a Send Characters.spin.
- ✓ Add the variable `i` to the list of long variables in the `VAR` block. It should read `long x, y, i`.
- ✓ Replace `pst.Str(@Win)` with this loop:

```
repeat i from 0 to 275
  pst.char(Win[i])
```

Project

- ✓ Try adding a "YOU LOSE" unhappy face to the program.

Learn More about DAT

- ✓ Look up `DAT` in the Propeller Manual's index.

Lesson 4: Pushbutton Monitoring

Pushbutton circuits are common in many appliances and inventions. This lesson introduces simple pushbutton circuits and some techniques for monitoring them. It also demonstrates how to use pushbutton state information to control LED light circuits. Of course, in place of the lights, your inventions might instead have circuits with on/off motors, heating elements or other devices, but the programming approach would be the same.
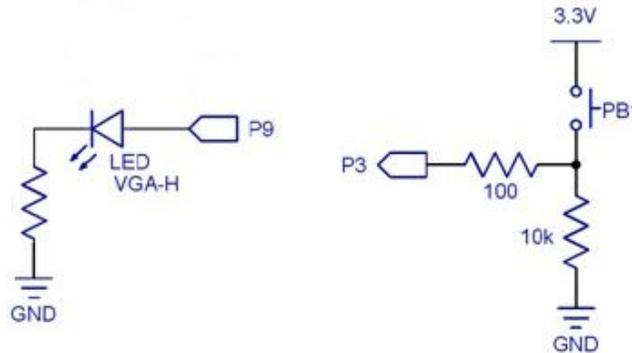
✓ Open the PropBOE_Spin_Buttons_20120130 folder.
✓ Follow these sections for a guided tour through each example program and how they work.

Pushbutton Circuit

This circuit uses two of the LED circuits from Lesson 2, as well as a couple of pushbutton circuits built onto the breadboard.
✓ Build the circuit shown in the schematic with the aid of the wiring diagram.

Remember to always move the 3-position switch to position 0 before building or changing circuits on your board!

Schematic
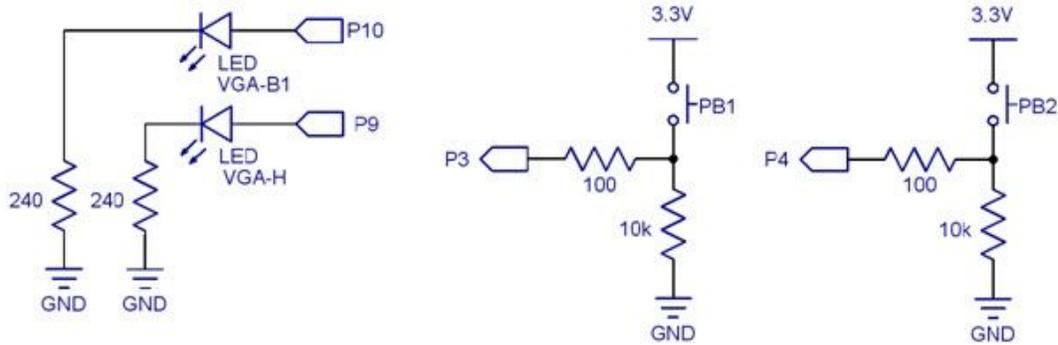
Wiring Diagram
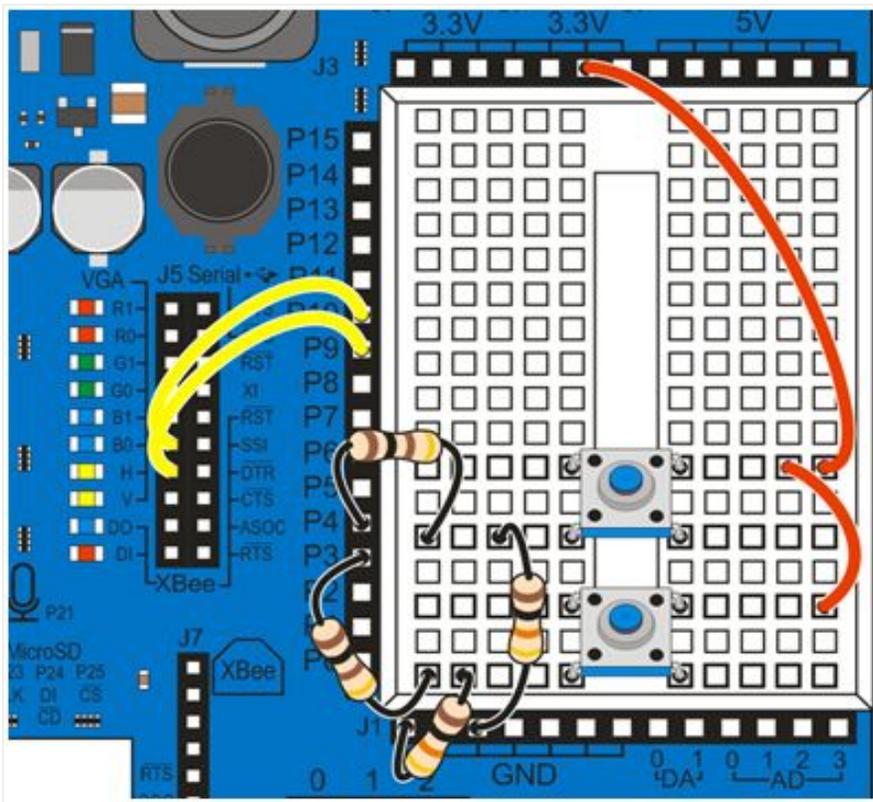
How it Works: Pushbutton Circuits

When the pushbutton is not pressed, the circuit applies GND (0 V) to the I/O pin. If the pushbutton is pressed, the circuit applies 3.3 V to the I/O pin, and a small amount of current passes from the 3.3 V connection, through the 10 kΩ resistor to ground. No current passes into P3 or P4 since they will be set to input by the programs.

Test a Pushbutton

This program displays the state of the pushbutton connected to P3. It should display 1 while the button is pressed, and 0 while it is not pressed.

- ✓ Open "1 Test Pushbutton.spin" and use F11 to load it into the Propeller chip.
- ✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
- ✓ Press and hold the button down. The display should show a 1.
- ✓ Release the button. The display should show a 0.
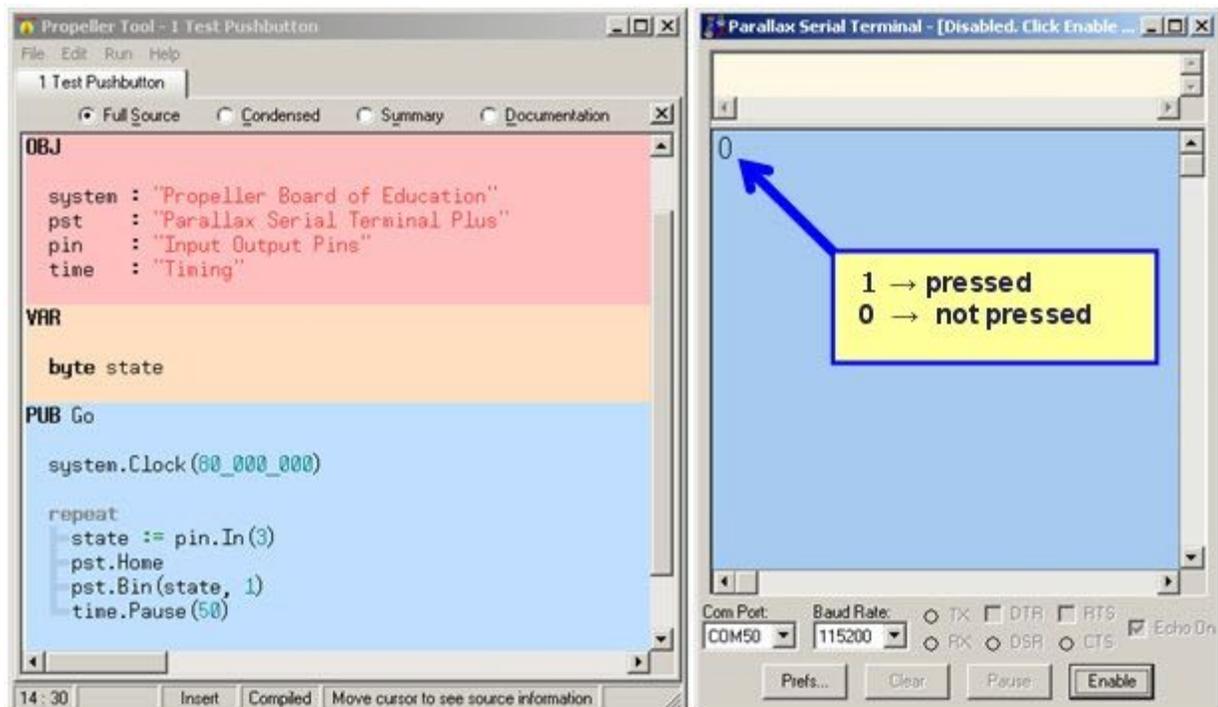
How it Works

Take a look at the `OBJ` block. The program is using library objects from both Lesson 1 and Lesson 2. The program also declares a variable named `state`. The expression `state := pin.In(3)` takes the value returned by the Input Output Pins object's `In` method and copies it to the `state` variable. The call to `pst.Home` places the cursor in the Parallax Serial Terminal's top-left home position. Then, `pst.Bin(state, 1)` displays the value of the `state` variable as a 1-digit binary number. Before repeating the loop, `time.Pause` eats up 50 ms, which is mainly to make sure that the program doesn't send messages faster than it needs to.

Your Turn

It's pretty easy to modify this example program to test the pushbutton connected to P4.

✓ Save the program under a new name, preferably starting with 1a. Like "1a Test Pushbutton.spin".
✓ Change `state := pin.In(3)` to `state := pin.In(4)`.
✓ Retest the program, this time pressing and releasing the pushbutton connected to P4.

Learn More about pin.In

Take a look at the `In` method documentation in the Input Output Pins object.
✓ In the Propeller Tool software, click Run -> Compile Current -> View Info… F8. (The F8 next to the menu selection means that you can use the F8 key as a shortcut to this feature.)
✓ Double-click Input Output Pins to open it. Then close the Object Info window.
✓ The Input Output Pins object should be the active tab in your Propeller Tool software now. Click the Documentation radio button.
✓ Read the In method's documentation. How does it relate to the example program?
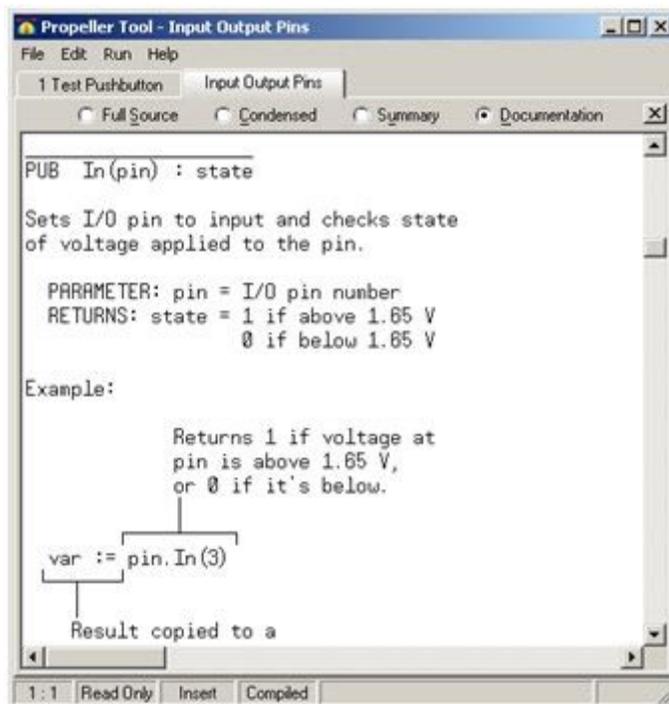
Control a Light with a Pushbutton

This example program blinks the LED light circuit connected to P9 while the P3 button is pressed and held. It will also blink the P10 LED light while the P4 button is pressed and held.
✓ Load the program into the Propeller chip with the F11 key.
✓ Press and hold the P3 pushbutton. Does the yellow LED labeled H blink?
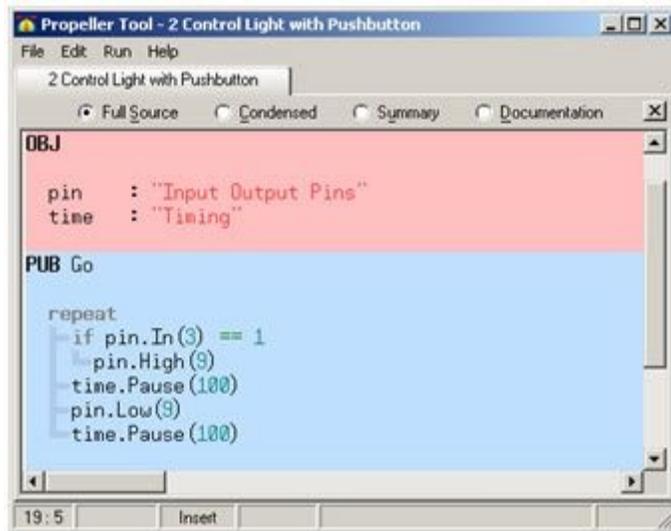
How it Works

Since `pin.In` returns either a 1 or a 0, why not just use it as the condition in an if command? Like this:

```
if pin.In(3) == 1
   pin.High(9)
```

Did you know?

The `:=` operator assigns a value or expression result on the right to a variable on the left. The `==` operator makes a comparison between the term on the left and the one on the right. If they are equal, it returns `TRUE` (a 32-bit value that's all 1s); otherwise, it returns `FALSE` (the number 0).

An `if` command will execute its block of code if its expression evaluates to anything that's not `FALSE`. In other words, if the result is not zero, it'll run the code. If it is zero, it skips the code.

Your Turn

Get creative; there are a lot of parameters to play with in this program. You can change the blink rate by adjusting `time.Pause`, the light by adjusting the `pin.High` and `pin.Low` parameters. You can even make the light blink while the button is not pressed by changing `if pin.In(3) == 1` to `if pin.In(3) == 0`.

✓ Try it!

Test Two Pushbuttons

Remember in Lesson 2 when you used the Input Output Pins object's Outs method to control several LEDs with one method call? Well, you can also monitor a group of pushbuttons with the Ins method to monitor a group of I/O pins, connected to pushbuttons in this case.

- ✓ Load "3 Test Two Pushbuttons.spin" into the Propeller chip with the F11 key.
- ✓ Remember to click the Parallax Serial Terminal's Enable button while the program is loading.
- ✓ The left digit displays the state of the P4 pushbutton and the right digit displays the state of the P3 pushbutton. Try the four possible combinations of pressed and not pressed and check the values you see in the Parallax Serial Terminal.
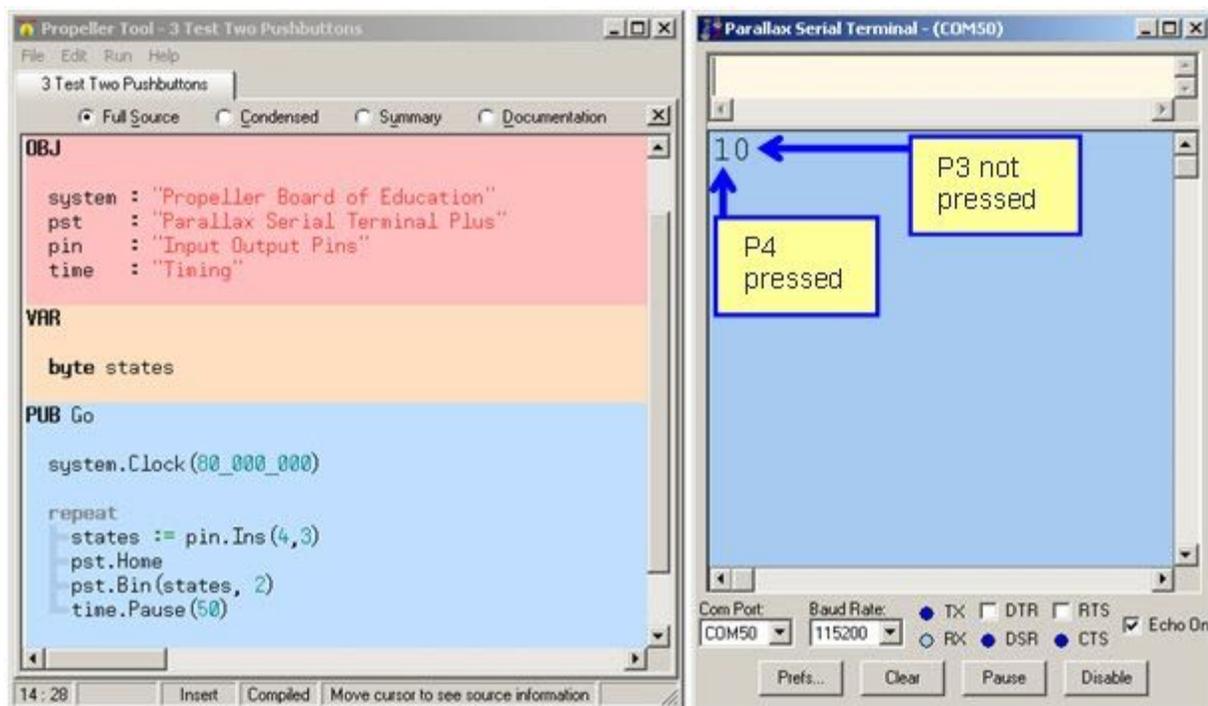
How it Works

The `pin.Ins` call makes the Input Output Pins object return a group of binary digits with 1s and 0s for each I/O pin. Since the call was `pin.Ins(4, 3)`, the call will return a 2-digit binary number with the P4 and P3 pin states.

The Parallax Serial Terminal's `Bin` method displays binary numbers in the Parallax Serial Terminal. The first parameter should contain the value to display, and the second one should contain the number of digits to display. Since we know that there are two useful binary digits in

the value the `pin.Ins(4, 3)` call returns, we can make the Parallax Serial Terminal display just those two digits with `pst.Bin(states, 2)`.

Your Turn

If you swap the 3 and the 4 in the `Ins` method call, what happens to the relationship between the digits displayed by the Parallax Serial Terminal and the buttons you press?
- ✓ Save "3 Test Two Pushbuttons.spin" as "3a Test Two Pushbuttons.spin".
- ✓ Swap the 3 and the 4 in the pin.Ins call, and retest to find out how the display responds to the buttons you press.

Learn More about Bin

- ✓ Press F8 to display the Object Info.
- ✓ Double-click the Parallax Serial Terminal Plus object to open it. Then close the Object Info window.
- ✓ The Parallax Serial Terminal Plus object should be the active tab in your Propeller Tool software now. Click the Documentation radio button.
- ✓ Read the `Bin` method's documentation.

Control Two Lights with Pushbuttons

This example uses two pushbuttons to control two LED lights.
- ✓ Open and run the program "4 Control LIghts with Pushbuttons." It'll load more quickly if you use F10 instead of F11.
- ✓ Press and hold each pushbutton and verify that its corresponding light blinks.

How it Works

The line `states := pin.Ins(4,3)` copies the binary values that indicate the states of the P4 and P3 pushbuttons to the `states` variable. Then, `pin.Outs(10, 9, states)` uses those values to apply high/low voltages to the corresponding LED light circuits. (High is 3.3 V, Low is GND = 0 V.) After a 0.1 second `time.Pause`, the `pin.Outs(10, 9, %00)` call turns both lights off. Another 0.1 second pause is required for the light to spend some time off as well as on to make it blink. Otherwise, the light would appear to just stay on while you press the button.

Did you know?

F10 is a shortcut for Run -> Compile Current -> Load RAM. EEPROM stores the program, even if you turn power off and back on or press/release your board's RST button. RAM does not.

EEPROM stands for Electrically Erasable Read Only Memory. RAM stands for Random Access Memory. When the Propeller chip starts up, if there isn't a computer trying to download a program to it, it copies the last program that was stored in EEPROM to RAM and then starts running it.

Your Turn

What happens without the last `time.Pause(100)`? The light blinks, but it spends such a short time off that the eye cannot detect it.
   ✓ Save a copy of your program with a new name.
   ✓ Place an apostrophe to the left of the last `time.Pause(100)` call, like this:
```
' time.Pause(100)
```

That removes it from the program, just like a code comment.

Expand Pushbutton Light Control

The program in the previous example isn't very flexible. It doesn't lend itself to adding extra code to do custom jobs depending on which button is pressed. Here is a program that uses a case command to do the same thing as the previous example. The difference here is that you can add more code to each case.
   ✓ Run Expand Pushbutton Light Control.spin and verify that it behaves the same as the previous example program.

```
Propeller Tool - 5 Expand Pushbutton Light Control
File  Edit  Run  Help
  5 Expand Pushbutton Light Control
    • Full Source    ○ Condensed    ○ Summary    ○ Documentation    ×

''Control P10..P9 LEDs with P4..P3 pushbuttons.
''Different appraoch with more options.

OBJ

  pin    : "Input Output Pins"
  time   : "Timing"

VAR

  byte states

PUB Go

  repeat
    states := pin.Ins(4, 3)
    case states
      %11: pin.Outs(10, 9, %11)
      %10: pin.Outs(10, 9, %10)
      %01: pin.Outs(10, 9, %01)
      %00: pin.Outs(10, 9, %00)
    pin.Outs(10, 9, %00)
    time.Pause(100)

1:1        Insert
```

How it Works

The case statement takes a look at the `states` variable, and checks for one of the four possible matches. When it finds a match, it executes the code that's either next to it, or below and indented from it. It only executes the case that matches the condition, then it skips to `pin.Outs(10, 9, %00)`.

Your Turn

Here is an example of a `case` statement that makes both lights blink alternately if both buttons are pressed. If the P4 button is pressed, the P10 light blinks, and if the P3 button is pressed, the P9 light blinks. If neither button is pressed, neither light blinks.
- ✓ Save your program under a new name (perhaps starting with 5a).
- ✓ Modify the `case` command as shown below.

```
PUB Go

  repeat
    states := pin.Ins(4, 3)
    case states
      %11:
        repeat 10
          pin.Outs(10, 9, %10)
          time.Pause(50)
          pin.Outs(10, 9, %01)
          time.Pause(50)
      %10:
        pin.Outs(10, 9, %10)
        time.Pause(500)
      %01: pin.Outs(10, 9, %01)
      %00: pin.Outs(10, 9, %00)
    pin.Outs(10, 9, %00)
    time.Pause(100)
```

Learn More about CASE

`Case` is a Spin command, so you can look it up in the Propeller Manual's Index and read lots more.

   ✓ In the Propeller Tool software, click Help and select Propeller Manual.
   ✓ Look it up in the index, and read up on it.

Lesson 5: Measure Voltage

Voltage is the electrical pressure that causes electric current to flow through a circuit. In this lesson, you will experiment with both measuring and generating voltages. The common names for these processes in electronics are *analog to digital conversion* and *digital to analog conversion*.
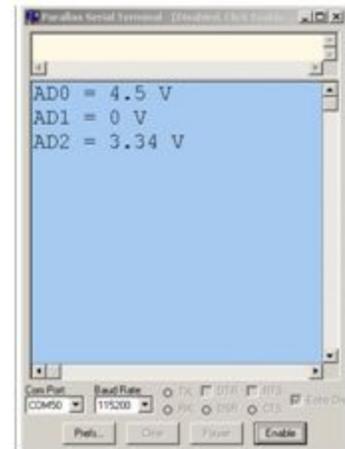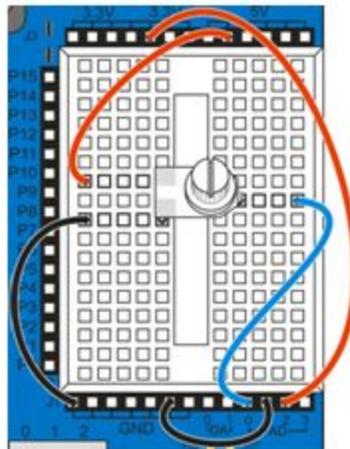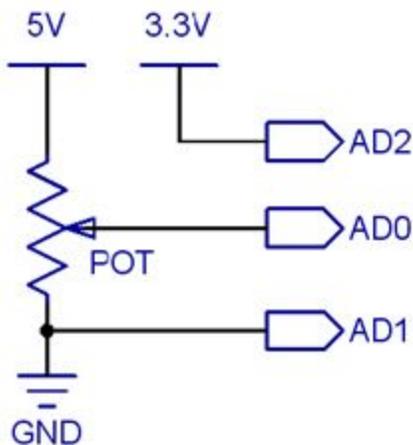
Several of the user library objects in the Voltage Measurements Spin Code package are works in progress. Please check the documentation comments for more info.
- ✓ Open the PropBOE_Spin_Voltage_20120613 folder.
- ✓ Start the lesson below.

ADC and Potentiometer Circuit

The Propeller Board of Education has four sockets that are connected to an onboard analog to digital converter chip. These sockets are labeled AD0, AD1, AD2, and AD3. In this activity, a potentiometer is connected to three of these sockets, as shown below.

As you turn the potentiometer knob, the voltage at the terminal connected to AD0 will vary. You can use the knob to adjust the voltage anywhere from 0 to 5 V. AD1 is connected to ground (GND), which should measure 0 V, and AD2 is connected to the 3.3 V socket.
- ✓ Use the schematic and wiring diagram to build the circuits.
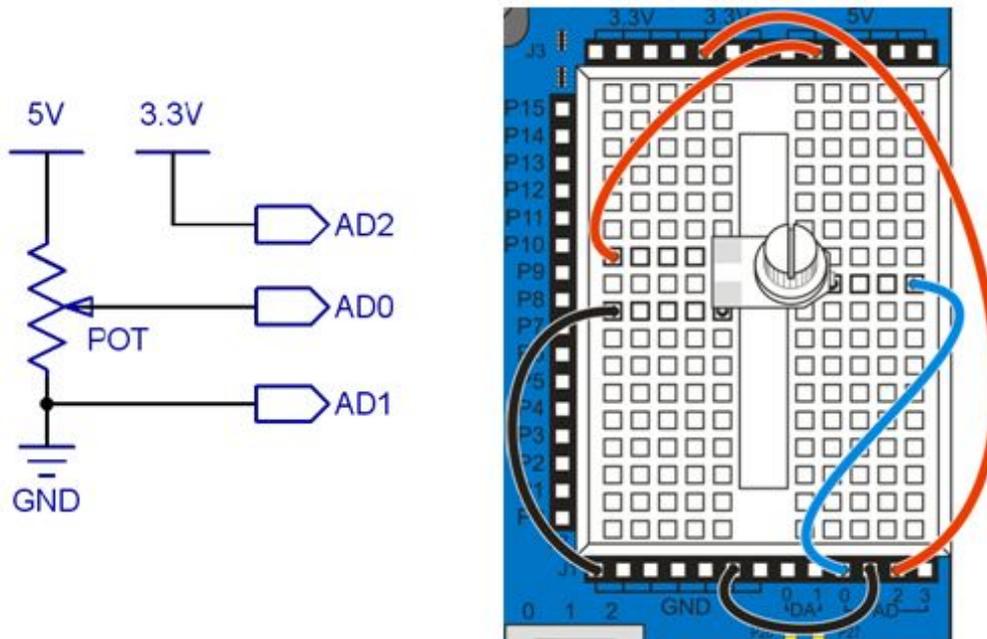
The potentiometer terminal connected to AD0 is called its wiper terminal. As you turn the knob, a contact connected to that terminal slides across a resistive element. 5 V is supplied at one end of the resistive element and 0 V is supplied to the other end. As you turn the knob one way or the other, the wiper slides across the resistive element, getting closer to either 5 V or 0 V.

Did you know?

Common abbreviations for *analog to digital conversion* are A/D conversion and ADC. Common abbreviations for digital to analog conversion are D/A conversion and DAC.

The value an A/D converter returns is an integer approximation of the actual voltage. It's called a *quantized measurement*.

Simple Analog to Digital Test

This first example displays the Propeller BOE's analog to digital measurement of the potentiometer's wiper terminal voltage output. The A/D converter will supply the Propeller with a value that corresponds to the voltage applied to AD0. The program stores this value in a variable, ad, and displays it in the Parallax Serial Terminal. The ad value is the number of 1024ths of 5 V. For example, in the figure below, the measurement is 927/1024ths of 5 V. That means the actual voltage is V(AD0) = 5 V × 927 ÷ 1024 ≈ 4.53 V.

- ✓ Open "1 Simple ADC Test.spin" and use F11 to load it into the Propeller chip.
- ✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
- ✓ Push the potentiometer down on the board as you adjust its knob so that it keeps electrical contact.
- ✓ Watch the Parallax Serial Terminal as you adjust the knob. The value of ad should vary from 0 to as high as 1023 as you turn the knob back and forth. It'll get larger as you turn the knob in one direction, and smaller as you turn it in the other direction.
- ✓ Try a few more voltage calculations with 5 V × ad ÷ 1024.
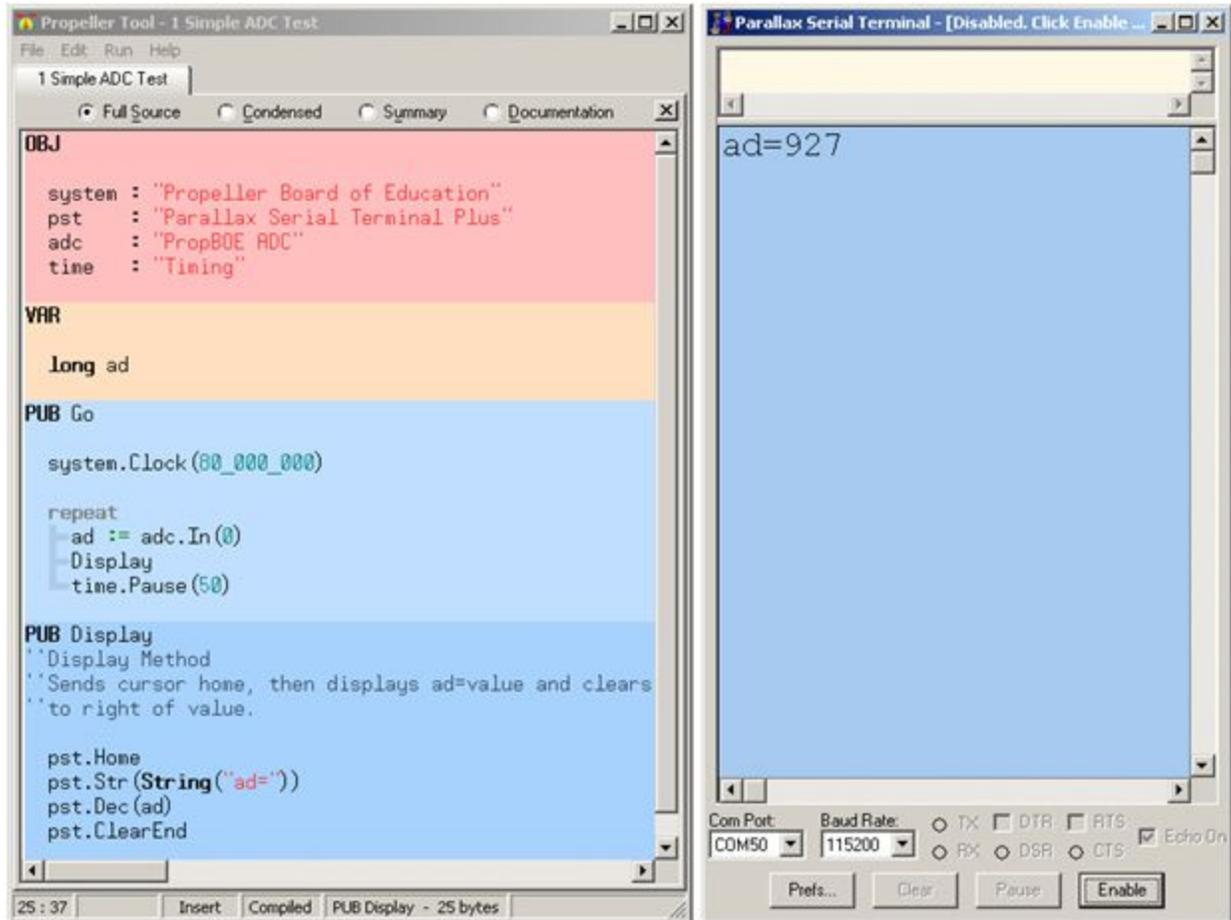
How it Works

The PropBOE `ADC` object's In method expects a channel from 0 to 3, and returns a value that the expression `ad := adc.In(0)` copies to a variable named `ad`. When the `In` method gets called, the code actually jumps into the PropBOE `ADC` object, finds its `In` method and starts executing it. When it runs out of commands in the method, it returns.

The `In` method's `PUB` declaration defines parameters the method needs to receive from the caller to do its job, as well as defining what value it returns. The call:

```
PUB In(channel) : adcval | pointer, acks, chan
```

...means that the In method expects one parameter named channel, and that it will return the value of a variable named `adcval` when it's done. The variables to the right of the | symbol are called *local variables*, and they use memory that the Propeller chip temporarily sets aside just while it's executing the method.

Receives channel parameter

Returns adcval

Source Code — 1 Simple ADC Test

```
OBJ

    system : "Propeller Board of Education"
    pst    : "Parallax Serial Terminal Plus"
    adc
    time

VAR

    long

PUB Go

    system.Clock(80_000_000)

    repeat
        ad := adc.In(0)
        Display
        time.Pause(50)

PUB Display

    pst.Home
    pst.Str(String("ad="))
    pst.Dec(ad)
    pst.ClearEnd
```

Call to method in other object passes 0 to channel parameter

Returns adcval

Call/return in same object. This method doesn't have any parameters, but it could

Source Code — PropBOE ADC

```
    long configured

OBJ

    i2c    : "I2C Slave 7bit"

PUB In(channel) : adcval | pointer, acks, chan
    ifnot configured
        i2c.Init(29, 28, %010_0011)

    pointer := |< (channel + 4)
    acks := i2c.poll(0)
    acks += i2c.ByteOut(pointer)
    acks += i2c.poll(1)
    adcval := i2c.ByteIn
    chan := adcval>>4
    adcval &= %1111
    adcval <<= 8
    i2c.AckOut(0)
    adcval += i2c.ByteIn
    adcval >>= 2
    i2c.AckOut(1)
    if acks <> 0 or chan <> channel
        adcval := -1

PRI Config(sda, scl, addr)
```

You can tell that `adc.In(0)` is a call to a method in another object because it starts with an object nickname from the `OBJ` block, adc in this case.

The `Display` method call below it calls another method, but it's not in another object, it's right below the `Go` method in the "1 Simple ADC Test" object. `Display` could also have one or more parameters and a return value, but this time it doesn't. When code in the `Go` method gets to the `Display` method call, it jumps down to `PUB Display` and starts executing code until it runs out, which could be the end of the file or the beginning of another `PUB` or `PRI` block.

Did you Know?

A method can have more than one parameter, but it only returns a single value. A method can be `PUB` (public) or `PRI` (private). A *public method* can be accessed by a call from another object. A *private method* can only be accessed within the same object. `PRI` is used when a

method helps other methods in the object do their jobs, but a call from outside the object might result in unpredictable behavior.

Your Turn

In the schematic, AD1 is connected go GND, 0 V. So a call to `adc.In(1)` should return 0. AD2 is connected to 3.3 V, so it should return a value in the neighborhood of `adc.In(2)` = 3.3 V × 1024 ÷ 5 V ≈ 676.
- ✓ Save the program under a new name, preferably starting with 1a. Like "1a Simple ADC Test.spin".
- ✓ Change `ad := adc.In(0)` to `ad := adc.In(1)` and verify that result, it should be 0. Repeat for `ad := adc.In(2)`.

Learn More about PUB

The Propeller Manual has lots more info on PUB/PRI method blocks.
- ✓ In the Propeller Tool software, click Help and select Propeller Manual.
- ✓ Look up `PUB` in the index, and read about it.

Multi Channel ADC Test

This program uses an indexing variable and a repeat loop to index through all three ADC channels. Remember that AD0 is connected to the potentiometer, so it should change as you twist its knob. AD1 is grounded, so `adc.In(1)` should return zero, and AD2 is connected to 3.3 V, so `adc.In(2)` should return about 676.
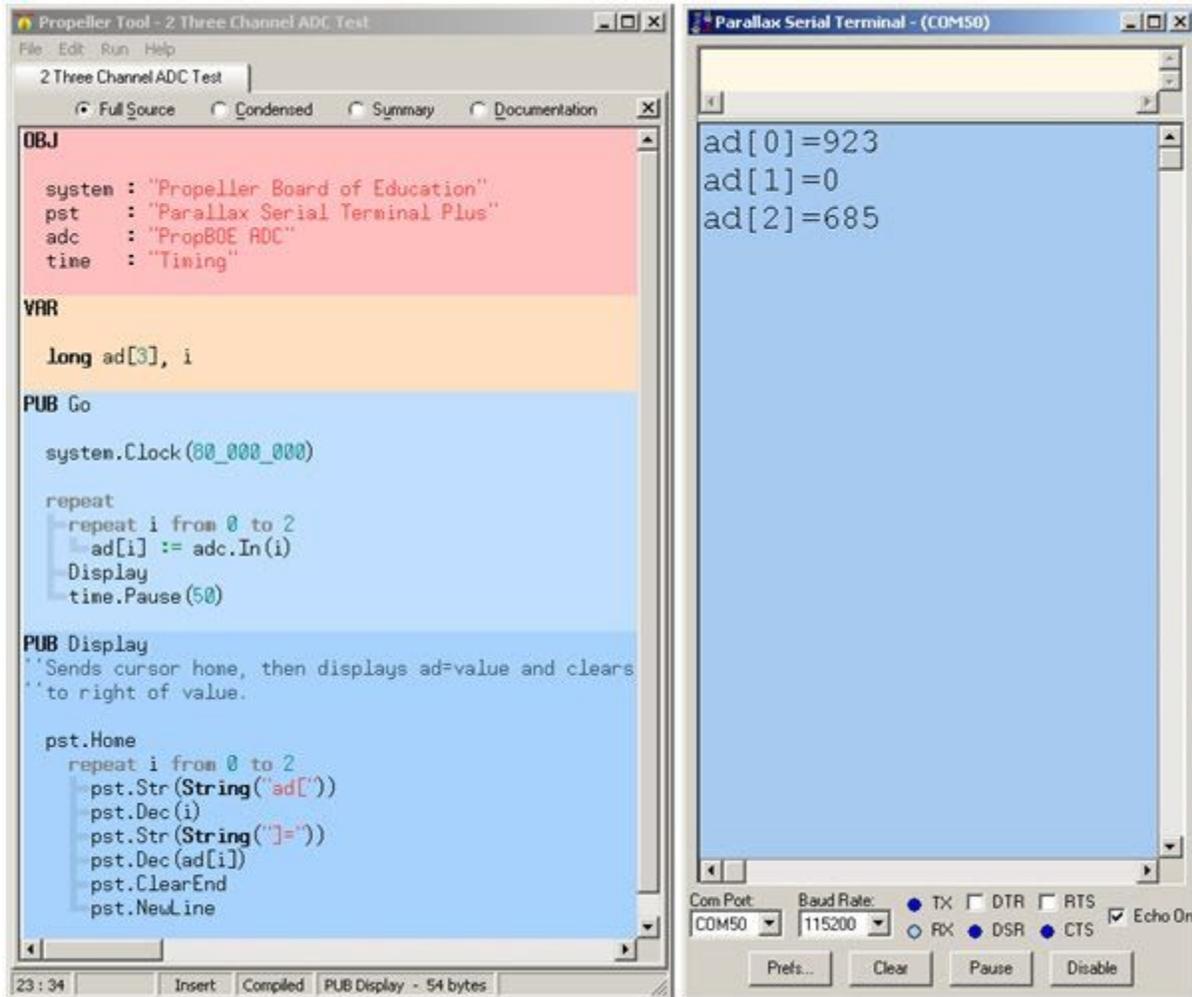
✓ Open "2 Three Channel ADC Test.spin" and use F11 to load it into the Propeller chip.
✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
✓ Verify that all three values display correctly.

How it Works

The loop:

```
repeat I from 0 to 2
   adc[i] := adc.In(i)
```

...repeats three times.

1. The first time through, i is 0, so `adc[i] := adc.In(i)` becomes `adc[0] := adc.In(0)`.
2. The second time through, i is 1, so `adc[i] := adc.In(i)` becomes `adc[1] := adc.In(1)`.
3. The third time through, i is 2, so so `adc[i] := adc.In(i)` becomes `adc[2] := adc.In(2)`.

. …and that's how it loads all three `adc` array variable elements.

Note that the `Display` method has been modified and uses a loop to display the various channel values.

Your Turn

You can display all three channels by changing `repeat i from 0 to 2` to `repeat i from 0 to 3`.
  ✓ Try it!

Voltmeter Test

Objects can be written for convenience. The `Voltmeter` object makes it so that you can use AD0…AD2 to probe your circuits and take voltmeter measurements. With voltage measurements, we'll expect to see the potentiometer's wiper voltage for AD0, 0 V for AD1 and a value that's pretty close to 3.30 V for AD2.
  ✓ Open "3 Voltmeter Test.spin" and use F11 to load it into the Propeller chip.
  ✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
  ✓ Verify the measurements.
  ✓ Slowly adjust the potentiometer's knob from one end of its range of motion to the other and keep an eye on AD0 as you do so.
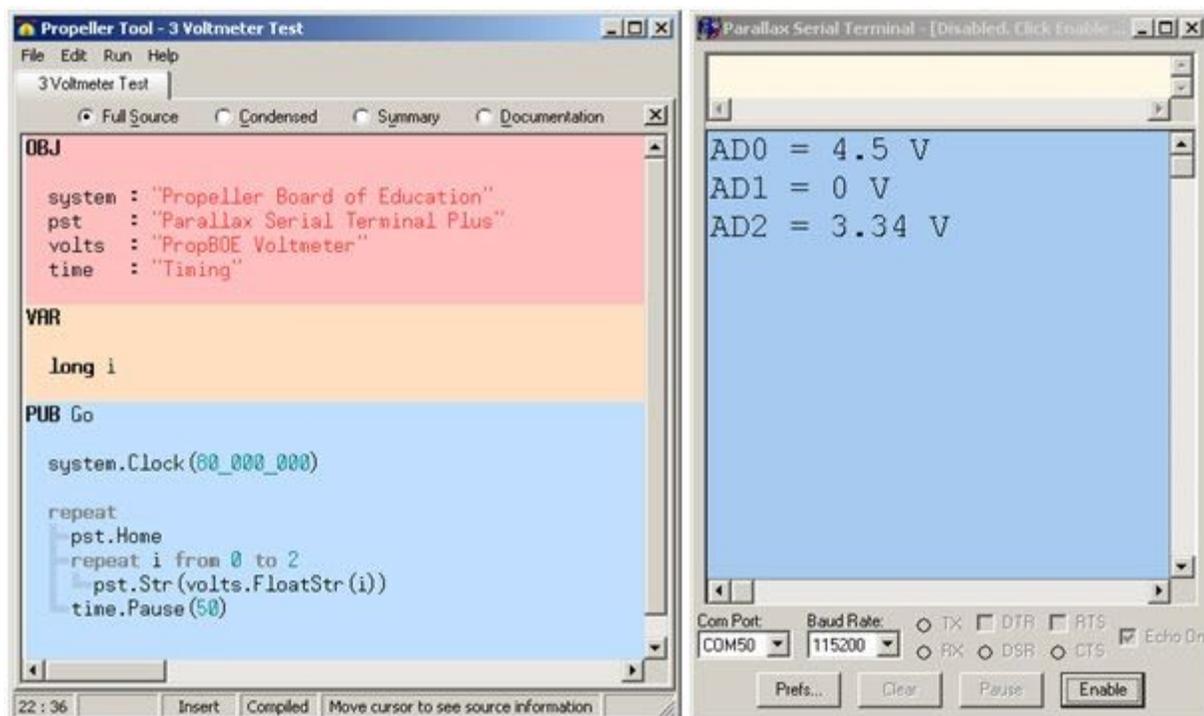
How it Works

Remember that the Parallax Serial Terminal's `Str` method expects the starting address of a zero-terminated string. The PropBOE `Voltmeter` object's `FloatStr` method returns the starting address of a zero-terminated string that contains the character representation of the floating-point voltage. At the end of the string are two ASCII control characters (11 and 13) followed by the zero terminator.

Did you Know?

ASCII control character (11) = Clear to End of Line. This helps avoid phantom characters when the new value has fewer characters than the previous one. ASCII control character (13) = New Line. In the Parallax Serial Terminal you can click Prefs -> Function to view the supported ASCII Control Characters.

Learn More about the PropBOE Voltmeter Object

PropBOE `Voltmeter` has methods that return the raw A/D integer value, a single precision floating point value, or the strings we experimented with here. The single-precision, floating-point value is useful for additional processing with the `FloatMath` object, which is in the Propeller Library.
- ✓ In the Propeller Tool software, click Run -> Compile Current -> View Info… (shortcut key F8).
- ✓ Double-click the PropBOE `Voltmeter` object to open it. Then close the Object Info window.
- ✓ The PropBOE `Voltmeter` object should be the active tab in your Propeller Tool software now. Click the Documentation radio button.
- ✓ Read the In method's documentation. How does it relate to the example program?

Simple Digital to Analog Test

Digital to Analog conversion is a reverse of the Analog to Digital Conversion process. While A/D conversion measures a voltage and spits out a number, D/A conversion takes a number, and generates the corresponding voltage.

On the Propeller Board of Education, the Propeller microcontroller's P26 and P27 I/O pins are connected to D/A converter circuitry that can take a number from 0 to 255 and generate a voltage from 0 to almost 3.3 V. (The values your program passes to `dac.Out` will be a number of 256ths of 3.3 V.) There are two LEDs below the DA0 and DA1 sockets that indicate the voltage. They get brighter with higher voltages and dimmer with lower ones. The sockets also

supply the D/A voltages so that you can connect them to circuits on your board, but we'll try that later.

✓ Don't worry about making any changes to your circuit, we'll just use the P26 and P27 voltage indicator lights below the DA0 and DA1 sockets for this activity.
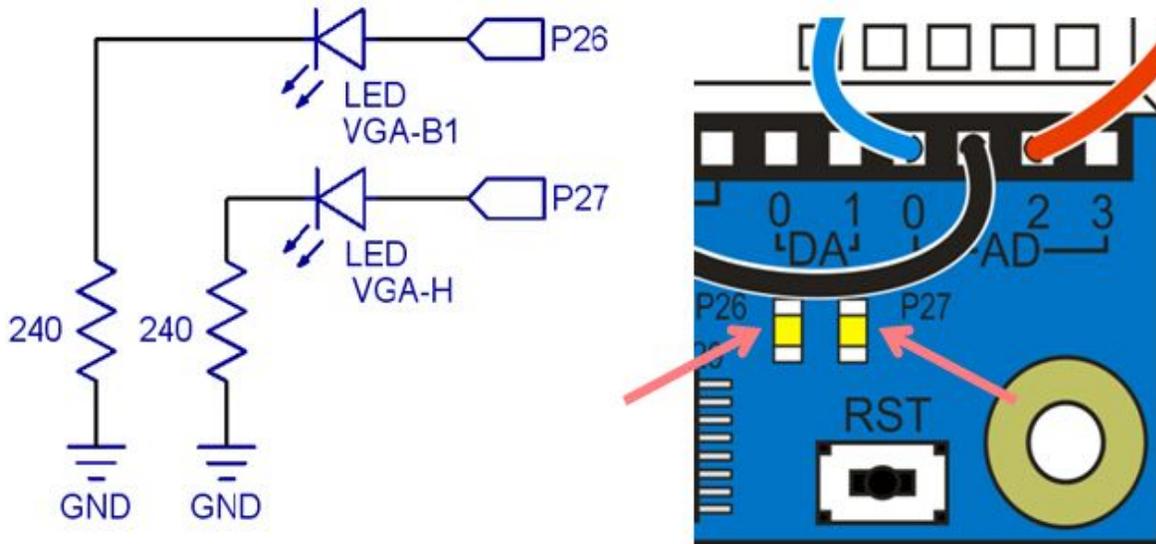
The "4 Simple DAC Test" object counts from 0 to 255 in steps of 4, and uses those values to set the DA0 and DA1 socket voltages.

✓ Open "4 Simple DAC Test" and use F11 to load it into the Propeller chip.
✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
✓ Verify that the P26 light gets brighter as DA[0] counts upward.
✓ Verify that the P27 light gets dimmer as DA[1] counts downward.
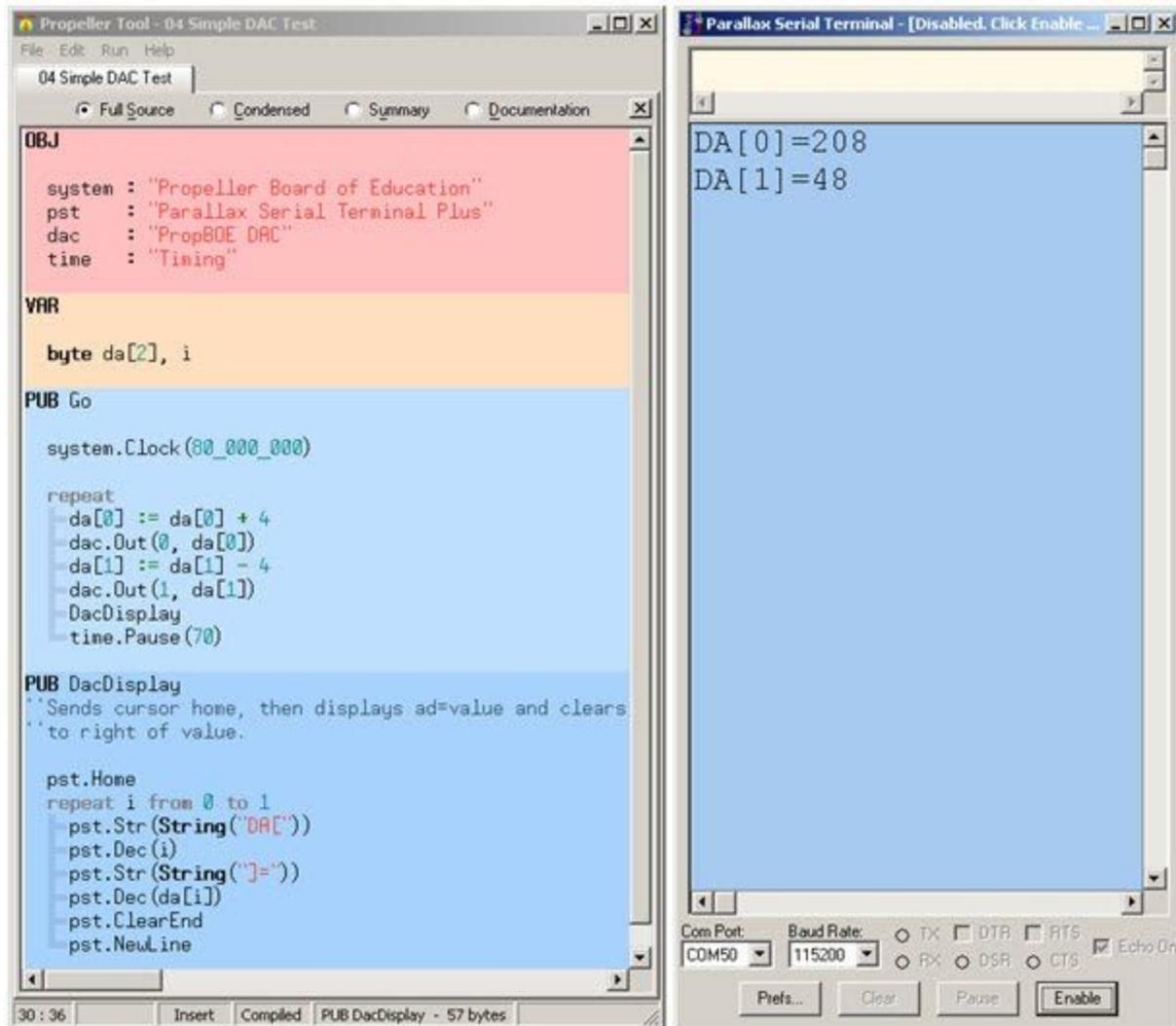
How it Works

The PropBOE `DAC` object is nicknamed `dac`, and its `Out` method expects two parameters, `channel` and `dacval`. A call to `dac.Out(0, 128)` sets the DA0 to 3.3 V × 128 ÷ 256 ≈ 1.65 V, and the P26 LED will be about half of full brightness. A call to `dac.Out(1, 64)` would set the DA1 socket voltage to 3.3 V × 64 ÷ 256 ≈ 0.825 V.

Instead of discrete values, the code uses `da[0] := da[0] + 4` and `da[1] := da[1] − 4` to make the `da[0]` array variable element count up in steps of 4 and `da[1]` count down in steps of 4. Every 70 or so milliseconds, the values stored in `da[0]` and `da[1]` are updated, and so are the `dac.Out` calls that set the DA0 and DA1 socket output voltages.

Did you know?

The assignment form of the add operator is +=, and you can use it to simplify some expressions. For example, you can substitute `da[0] += 4` in place of `da[0] := da[0] + 4`. You can even replace the first two commands in the `repeat` loop with `dac.Out(0, da[0]+=4)`.

Your Turn

The four lines after the `system.Clock` call each set voltages at DA0 and DA1. The fourth line is a `repeat` loop with nothing below or indented from it. So the code gets stuck there and does nothing else.
- ✓ Save the program under a new name, something like "4a Simple DAC Test.spin".
- ✓ Add the four lines of code shown below.
- ✓ Run the code. It should set the P27 LED below DA1 to ¾ brightness and the P26 LED below DA0 to ¼ brightness.

```
PUB Go

  system.Clock(80_000_000)

  dac.Out(0, 64)          ' DA0 = 0.835 V, P26 LED dim
  dac.Out(1, 192)         ' DA1 = 2.475 V, P27 LED brighter
  pst.Str(String("Check P26 & P27 LEDs"))
  repeat                  ' code stops here
                          '
```

Add
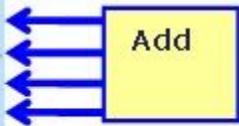
Let's also try the code technique using +=, shown in the Did You Know section.
- ✓ Reopen "4 Simple DAC Test.spin" and save it as "4b Simple DAC Test.spin".
- ✓ Comment the four lines shown below by placing an apostrophe to the left of each one.
- ✓ Add the two lines (each replaces a pair of lines that you commented).
- ✓ Run the code. The lights should still do the same bright/dim pattern the original one gave you.

```
PUB Go

  system.Clock(80_000_000)

  repeat
    'da[0] := da[0] + 4
    'dac.Out(0, da[0])
    'da[1] := da[1] - 4
    'dac.Out(1, da[1])
    dac.Out(0, da[0]+=4)
    dac.Out(1, da[1]-=4)
    DacDisplay
    time.Pause(70)
```

Comment with apostrophes left of each

Add
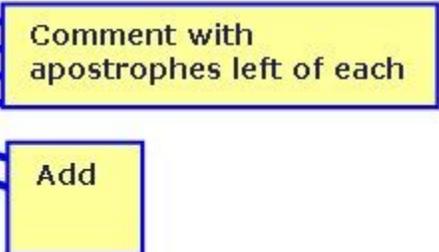
## Learn More about Operators: + vs +=

The Propeller Manual explains the difference between using + and +=.
- ✓ In the Propeller Tool software, click Help and select Propeller Manual.
- ✓ Look up binary operators in the index.
- ✓ Follow the reference to the page that explains + and +=. It explains the operator in its binary + and assignment += forms.

Monitor DAC with Voltmeter

We can even use the AD inputs to measure the DA outputs.
- ● Connect AD1 to DA0 and AD2 to DA1 as shown in the schematic and wiring diagram.
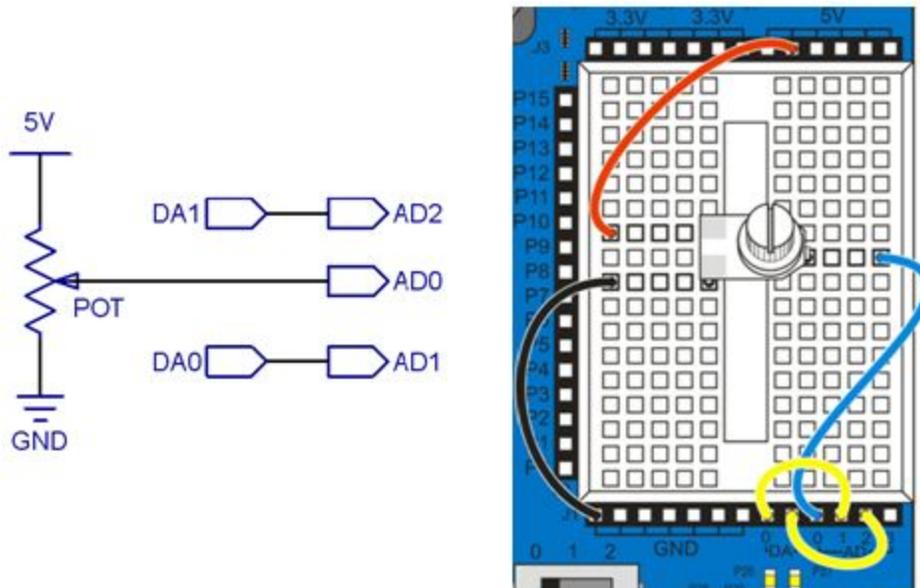
The program 5 Monitor DAC with Voltmeter.spin does the same up/down count to set the DA0 and DA1 socket voltages and make the P26 and P27 LEDs get brighter and dimmer. It also uses the Voltmeter object to monitor all three AD channels, so you can monitor the potentiometer voltage along with the two DA voltage outputs.
- ✓ Open "5 Monitor DAC with Voltmeter.spin" and use F11 to load it into the Propeller chip.
- ✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
- ✓ Verify that AD1 increases from 0 to 3.3 V as AD2 decreases from 3.3 V to 0 V.
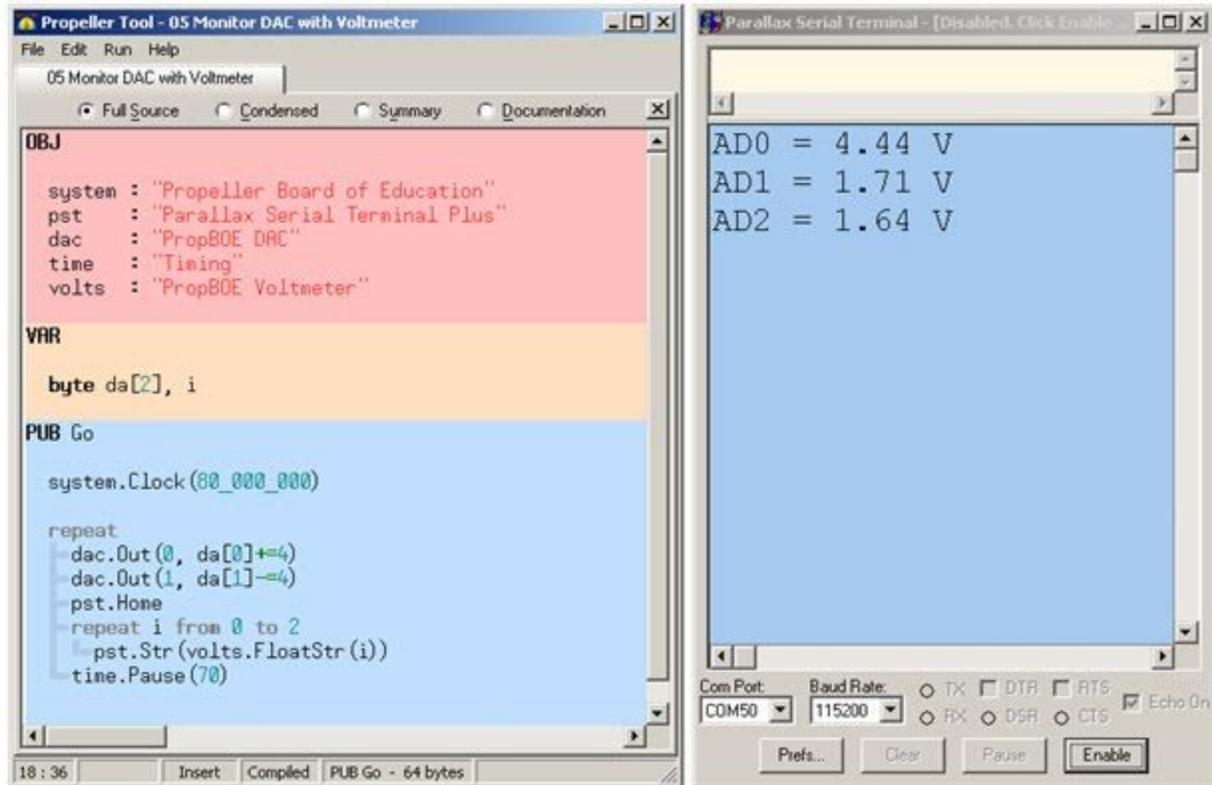
Monitor DAC with Oscilloscope

The dancing voltage measurements from the previous activity might not be as helpful as a graph. We can use the PST Scope object to graph the AD1 and AD2 measurements.
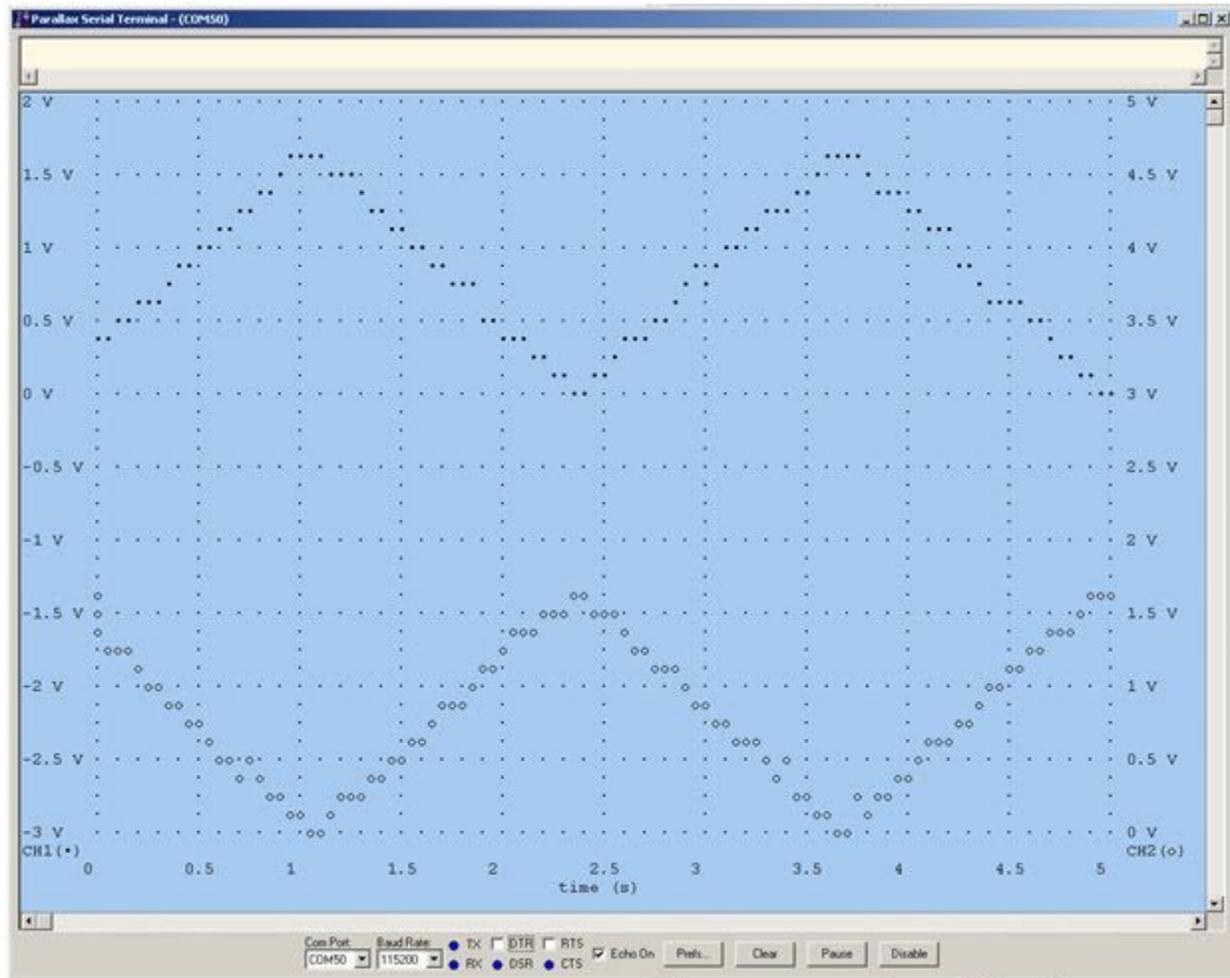
The Parallax Serial Terminal needs to be configured to display the scope data first.

- ✓ Click the Parallax Serial Terminal's Prefs... button.
- ✓ In the Appearance tab, set the Font size to 10, Wrap text to Page, and Page width (characters) to 128.
- ✓ Click the OK button to exit the Preferences window.
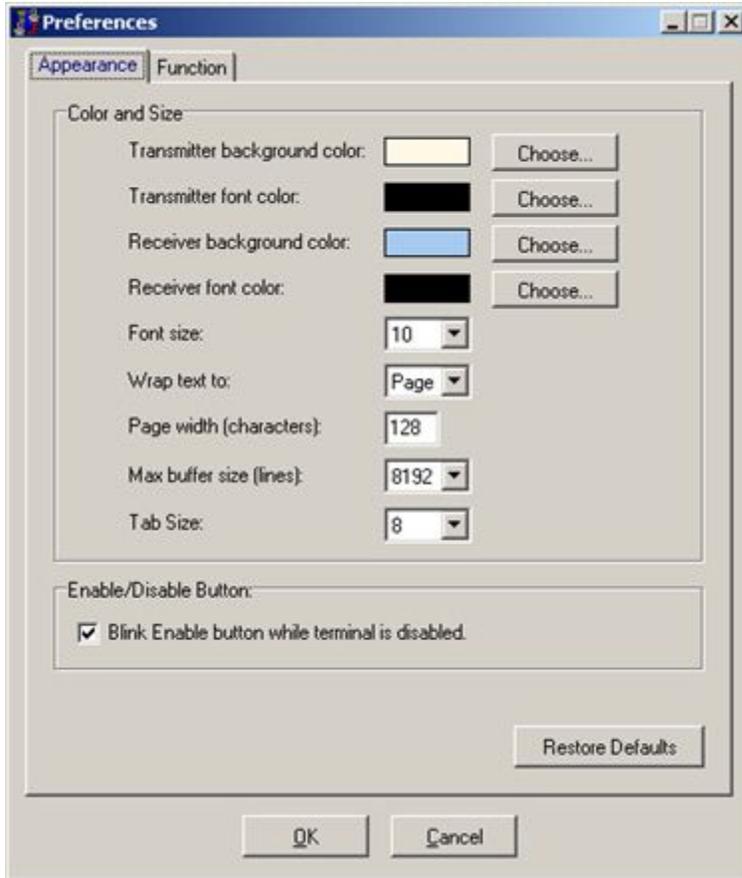
Now, let's test the PST Scope...

✓ Open "06 Test PSTscope.spin" and use F11 to load it into the Propeller chip.
✓ While the program is loading, click the Parallax Serial Terminal's Enable button.
✓ You will need to wait for several P26/P27 LED bright/dim cycles before the graph of the increasing/decreasing voltages appears.

How it Works

The PST Scope object requires a `DAT` block with all the oscilloscope settings shown in the order that they are shown. Then, its `Start` method requires the starting memory address of those values, which is `@Scales`.

The PST Scope object takes over all the display updating, and your test code can focus on generating a signal. You can use the Parallax Serial Terminal to watch the signal graphically, and verify that it is what you intended to create.

Did you know?

Most objects that start a process in another cog and manage it for you have a `Start` method you have to call. They also usually have a `Stop` method so that you can recover the cog for use with another process. The PropBOE library is just the beginning. There are lots of objects that perform useful tasks and make connecting peripherals to your propeller a snap. Two big object repositories:

1. The Propeller Object Exchange at http://obex.parallax.com [6].
2. The Propeller Library in the Propeller Tool software (File -> Open From -> Propeller Library).

Your Turn

You can adjust the rate of the D/A signals by adjusting the time.pause method call. For example, you can double the signal's frequency (number of times it repeats per second) by cutting the time.pause durations in half.

✓ Change both instances of `time.Pause(10)` to `time.Pause(5).`
✓ Load the modified program into the Propeller.
✓ Take a look at the new signal in the Parallax Serial Terminal. Since it's repeating twice as fast, there should be twice as many repetitions visible.

If you make the signal repeat even more quickly, it might not display very well. But, you can fix that by reducing scale on the time axis. For example, you could reduce both `time.Pause(5)` to `time.Pause(2)` and then make the width of each square in the graph represent 0.2 seconds instead of 0.5 seconds.

✓ Change both instances of `time.Pause(5)` to `time.Pause(2).`
✓ Load the modified program into the Propeller, and click the Parallax Serial Terminal's Enable button. Check the display; it won't look very good.
✓ Decrease the time scale by changing the number below the t(s/div) comment from 0.5 to 0.2. (It's in the `DAT` block.)
✓ Load the modified program into the Propeller and click the Parallax Serial Terminal's Enable button.
✓ Check the display again, it should look considerably better.
✓

Both signals swing between 0 V and 1.65 V. In oscilloscope-speak, you could say that each signal has a "peak-to-peak amplitude of 1.65 V". If you were to double the CH1 signal's amplitude by making it swing from 0 to 3.3 V, it wouldn't fit in the display any more. To fix that, you could adjust the channel 1 axis scale. Let's try it.

✓ Double the signal amplitude the `DAC` sends to channel 1 by changing both `dac.Out(0, da)` calls to `dac.Out(0, 2*da).`
✓ Load the modified program into the Propeller and then click the Parallax Serial Terminal's Enable button.

✓ Check the LEDs by DA0 and DA1 on your Propeller BOE. The DA0 LED should now get twice as bright as the DA1 LED.
✓ At this point, the Channel 1 display does not show the whole signal. You can fix this by adjusting the value below ch1(V/div) label in the `DAT` block. Change it from 0.5 to 1.0.
✓ Load the modified program into the Propeller and verify that you can now see the entire signal in the display.
✓ Take a close look at the display. The two signals look the same height, but CH1 axis on the left indicates that the CH1 signal swings from 0 to 3.3 V while the CH2 axis on the right indicates that the CH2 signal still only swings between 0 V and 1.65 V.

At this point, the display probably still doesn't look quite right because the CH1 signal is now running into the CH2 signal. To move it up a little bit, you can adjust the CH1 axis' offset from the bottom of the screen.
✓ In the `DAT` block, change the value below the CH1(V) label from -3.0 to -4.0.
✓ Load the modified program into the Propeller, and click the Parallax Serial Terminal's Enable button
✓ Verify that the CH1 signal now displays up above the CH2 signal.