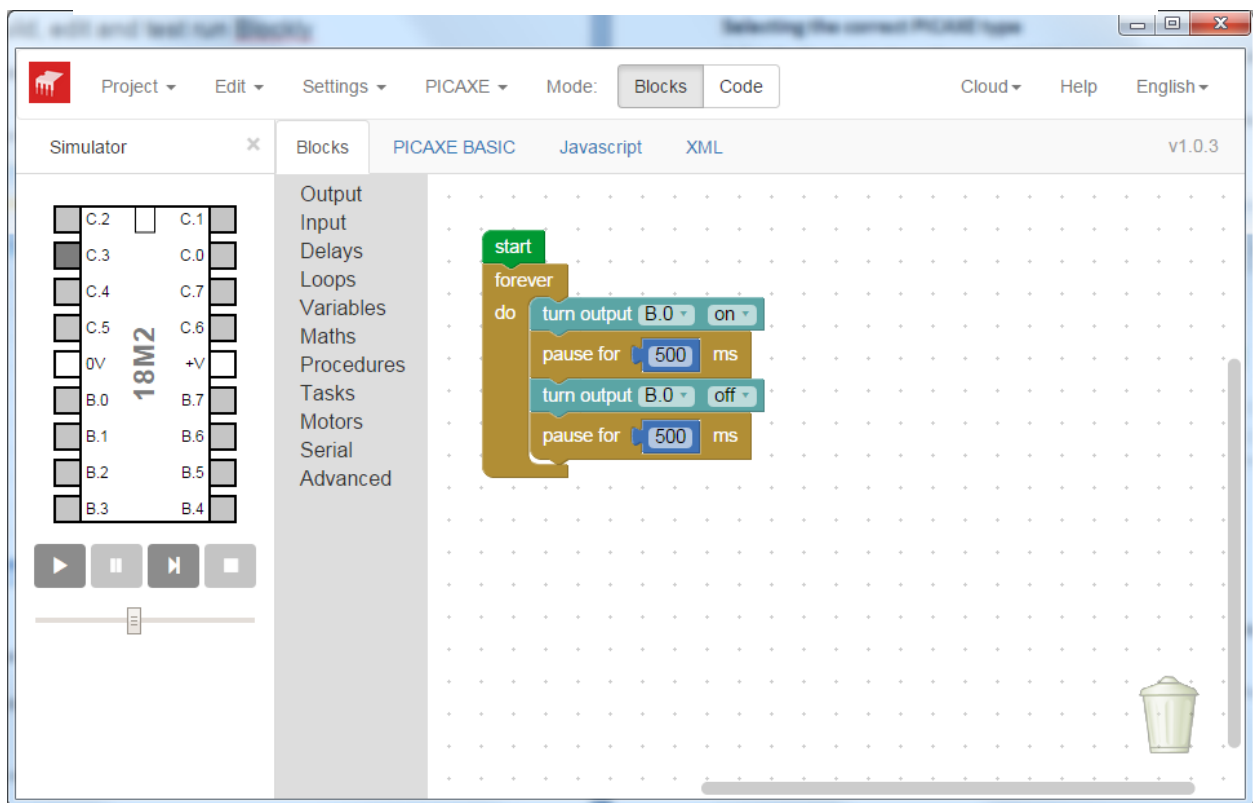


# A guide to using Blockly to simulate and program a PICAXE microcontroller



© Copyright Revolution Education Ltd 1999-2016.

Copyright is waived in the following circumstances: a small number of copies may be made for use in the purchaser's school/college for use alongside PICAXE hardware. These copies may not be sold or made available outside the purchaser's school.

## Overview

**Blockly for PICAXE** is a free powerful visual programming tool for generating PICAXE microcontroller programs. By stacking coloured blocks on top of each other a control program can be rapidly generated. This simple click'n'drag programming method allows students to rapidly develop control sequences for real life microcontroller projects.

Blockly can be run in a web browser on almost any device that has an internet connection. For offline use Blockly is embedded within PICAXE Editor and is also available as a standalone Chrome app.

*Blockly can be used in 3 different ways:*

- 1) Within PE6 (PICAXE Editor 6) which is the main PICAXE programming environment (Windows)*
- 2) As a standalone Chrome app (Windows/Mac/Linux/Chromebook)*
- 3) On the [www.picaxecloud.com](http://www.picaxecloud.com) website (any browser)*

*All 3 methods share exactly the same Blockly source code and so work in a similar way on all platforms. However PE6 does contain a more powerful simulation engine.*

The wide range of PICAXE specific blocks allows the user to control output devices, such as motors and LEDs that are connected to the PICAXE microcontroller. We can switch devices on or off in sequences using: timing, counting, repetition, and decisions based on signals from digital and analogue sensors that are connected to the PICAXE microcontroller.

This section of the manual explains how the most common blocks are used, giving examples of the common blocks and techniques in the context of possible school projects.

## Quick Start

If you are unfamiliar with the program approach to building control systems, it is a good idea to begin by familiarising yourself with the most commonly used blocks, which are: *Outputs, Wait, Motor* and *Inputs*.

## **1. How to build, edit and test run a program**

## **2. Outputs**

This section shows: how to switch output devices and motors connected to outputs of a PICAXE microcontroller, using *Outputs*, *Motor*, *Sound* and *Play* blocks; how the *Serout* block can be used to output serial information from the PICAXE microcontroller.

## **3. Inputs**

This section shows: how to check the state of digital sensors connected to a PICAXE microcontroller using the *input* block; how to use the *Interrupt* block for instant response to digital sensors; how to use the *variable decision* block to make use of readings from analogue sensors connected to a PICAXE microcontroller, in a control system.

## **4. Delays**

This section shows: how to create delays using *pause* and *sleep*

## **5. Procedures**

This section shows the important technique of building a control system as a number of linked sub systems.

## **6. Maths & Variables**

This section shows: how to create counting systems using *Increase* and *Decrease* blocks; how timing can be built into a control system; how *Expressions* and *Random* blocks are used to give a value to a variable; how *Read* and *Write* blocks are used to store and access values of variables using the PICAXE microcontroller's EEPROM memory.

## **7. Advanced Blocks**

This section shows: how to use some of the more advanced PICAXE command blocks.

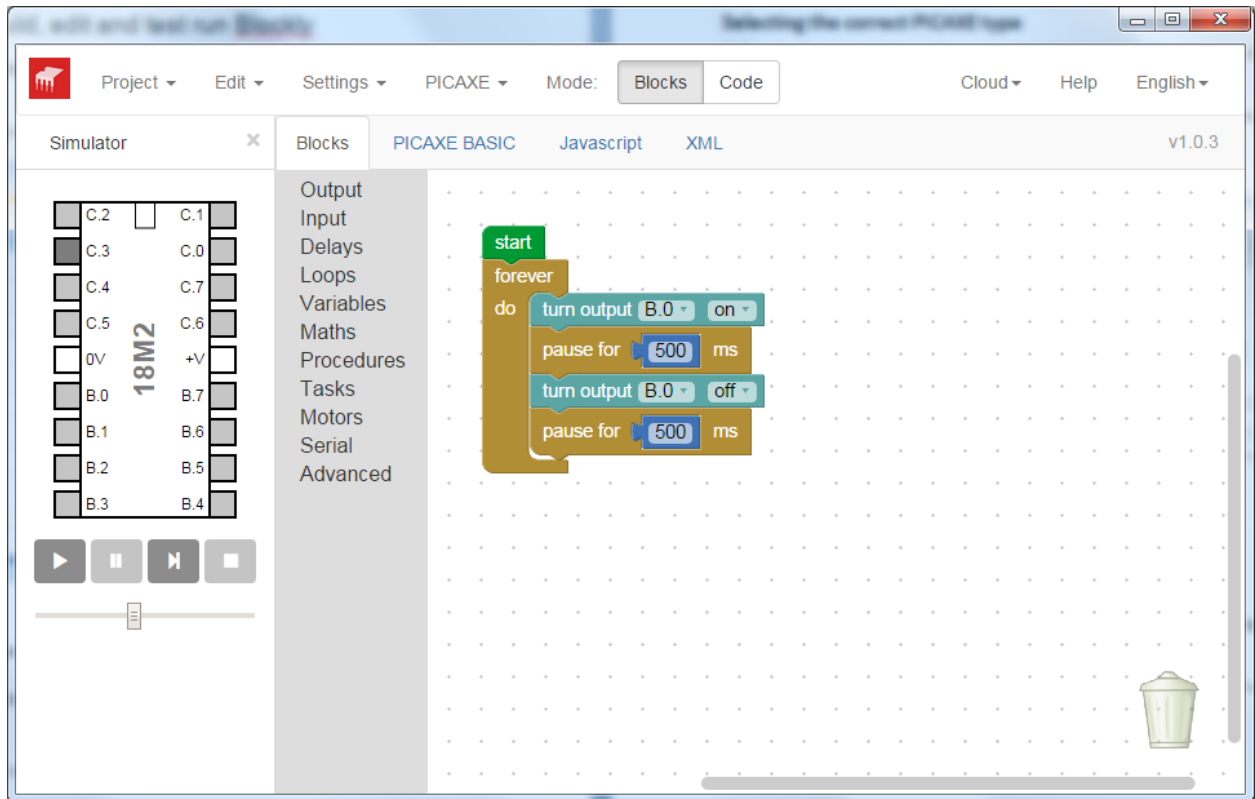
## Section 1. How to build, edit and simulate in Blockly

PE6 - Click the 'New Blockly' ribbon button.

App - Click Project > New

Web - Login and then click Project>New

The Blockly screen looks similar to:



Workspace - This is the right hand area where your program is created

Toolbox - This is the collection of available blocks to drag onto the stage

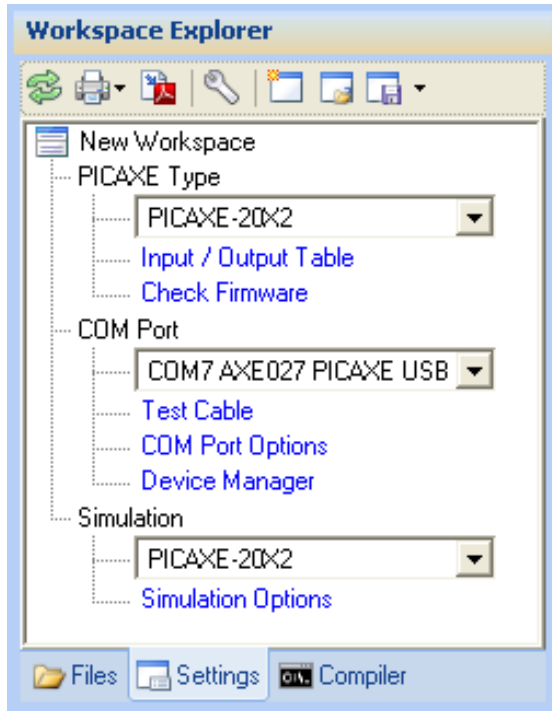
Simulation Panel –

This displays the animated simulation when the program is run 'on-screen', In PE6 the Simulation Panel looks slightly different, but performs the same task.

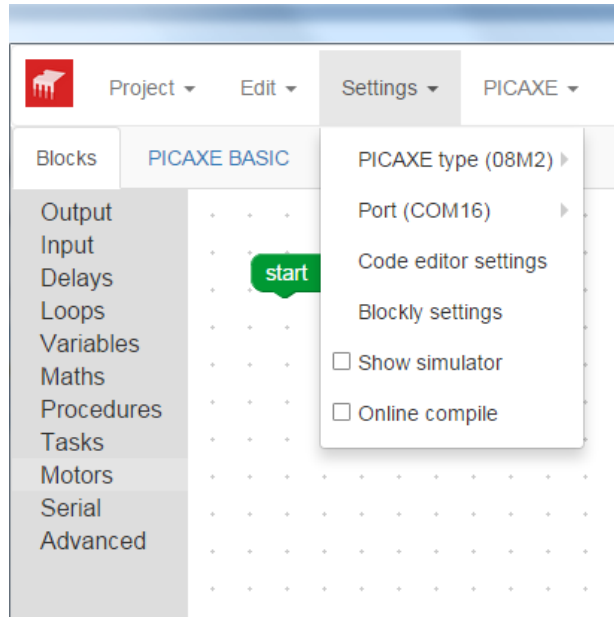
## Selecting the correct PICAXE type

Before the program is created the correct PICAXE microcontroller chip type and download cable COM port should be selected.

*PE6 – Use the Workspace Explorer*



*App – Use the Settings menu*

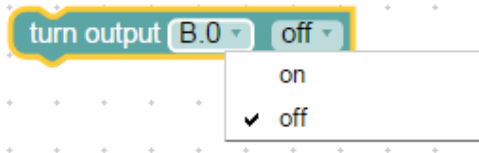


Note that if you have the wrong PICAXE chip selected the available input/output pins displayed in the block drop-down lists will not be accurate.

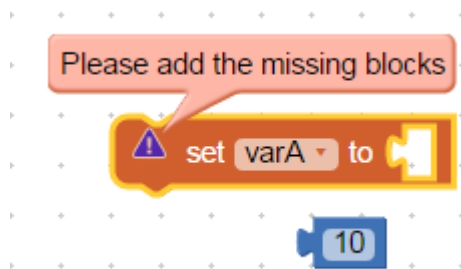
*NOTE: This section deals only with drawing the program. Details of how to use the individual blocks are given later.*

### Adding a new block

Drag the required block from the toolbox and place it on the workspace. Most blocks have a drop down list of options that are used to alter the way the block operates.

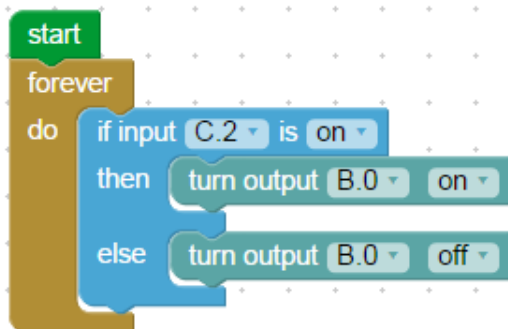


Some other blocks have a 'jigsaw piece' input position where another block can be dropped, for instance you may drop a constant (number) or a variable into this block.



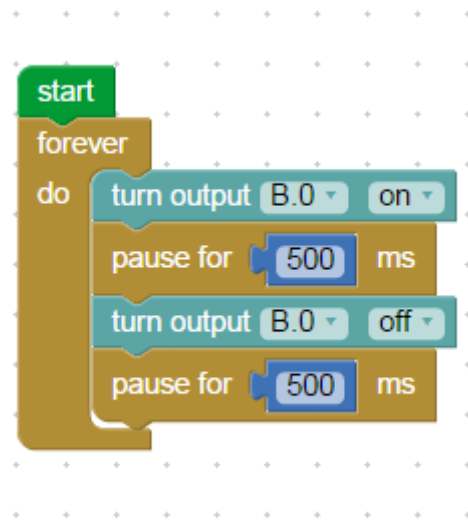
If a block is missing the '!' icon and warning may be shown, this warning will automatically disappear when the block is inserted.

Loops and decision blocks also allow other blocks to be stacked inside them e.g.



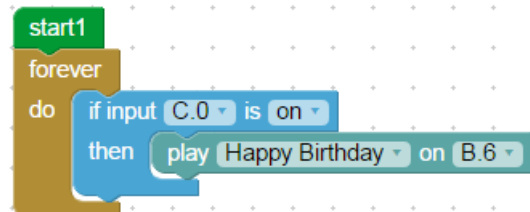
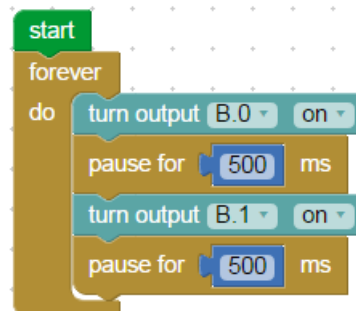
### Start block

A Start block marks the point where the program starts running.



When the PICAXE microcontroller is reset or powered up, the program starts at the first Start block. Every program must have at least one Start block. A program will stop running whenever a Stop block is reached.






For PICAXE-M2 parts you can have up to 8 Start blocks on each program. New Start blocks are found in the 'Tasks' toolbox section.



## Moving blocks

To move a single block or a stack of blocks, select the top block and drag it to its new position.

## Zooming & Deleting

-  At the bottom right hand side of the Blockly screen are 4 icons:
-  1) Zoom to 100% at centre
-  2) Zoom In
-  3) Zoom Out
-  4) Delete

To delete a block either

- 1) drag it into the 'trash can'
- 2) press the Delete key on the keyboard
- 3) right click and select 'Delete Block'

Note that as all programs need a Start block the first start block cannot be deleted.

## Cutting, Copying and Pasting

Use the Cut, Copy and Paste options from the Edit menu to cut or copy selected blocks or stack of blocks and paste them either into another part of the same program or into a different program.

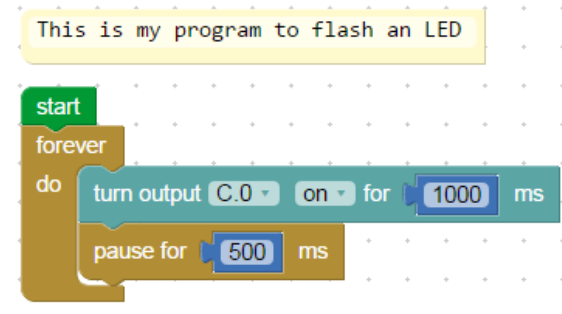
Alternatively, you can right click and select 'Duplicate Blocks'

## Grid

The grid can be hidden or displayed via the Blockly Settings. When the grid is displayed blocks automatically 'snap' to the nearest grid point.

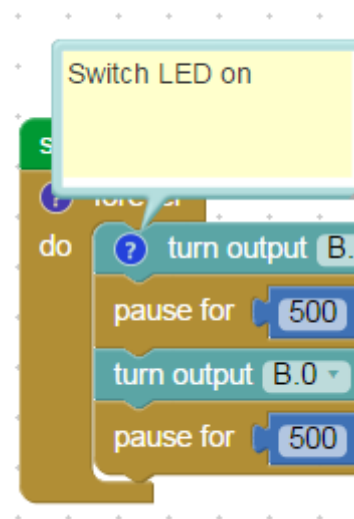
## Comment Blocks

It can be useful to drop comments onto your program to tell other people how it functions. Comment blocks are found in the 'Advanced' section.



## Labelling a block

It can be useful to give a block an internal comment which identifies what it is used for, e.g. "switch LED on". To add a comment right click over the block and select 'Add Comment'



A new '?' icon will then appear, when you click the icon the comment will be displayed and can be edited.

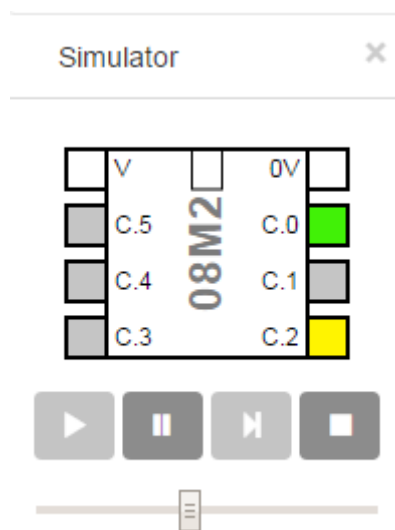
The block comment does not affect the operation of a block; they are only a label for 'humans' to read.

## How to test run a program

Before you download a program to a PICAXE microcontroller, it is useful to be able to check that it works as you intend it to. Simulation has a number of features that allow you to test run the program in the software.

### 1. The Simulation Panel

As a program runs, the Digital Panel shows the changing state of outputs and inputs as they would be if the program had been downloaded to a PICAXE microcontroller.

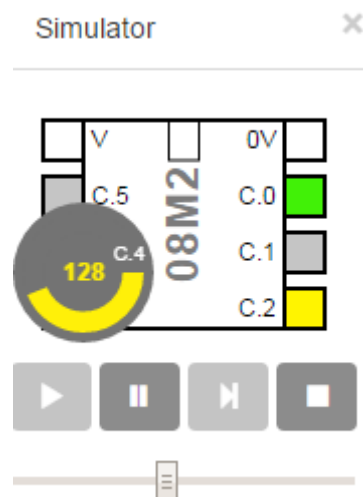


### 2. Simulating digital inputs

To change the state of an input simply click on the input in the simulation panel. It will turn from grey (off) to yellow (on).

### 3. Simulating analogue inputs

To change the value of an analogue input right click on the input pin to display the radial slider. Turn the slider as required.



### 4. Run and Stop

To test run a program click the Run button on the toolbar or press <Ctrl>+<F5>

To stop a program running click the Stop icon.

As the program runs, the flow of control is highlighted so that you can follow it. If you want to slow down the speed at which flow is highlighted is controlled by the simulation delay slider.

### 5. Breakpoints

Right click on a block to add a breakpoint flag to it. When the simulation reaches this point the program will then pause.

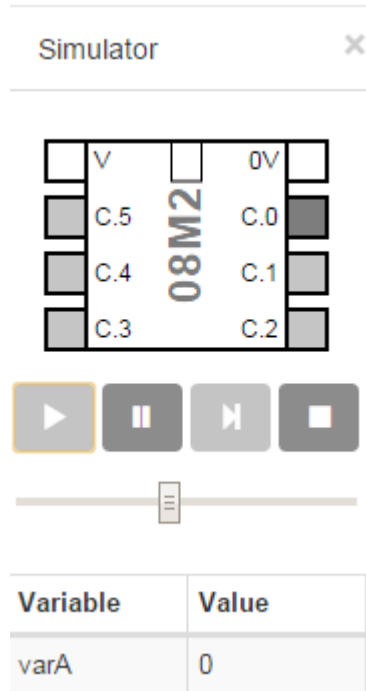




## 6. Variables display

If your program uses variables, it is useful to see the changing values of any of the variables that are used in the program will be displayed as the program runs.

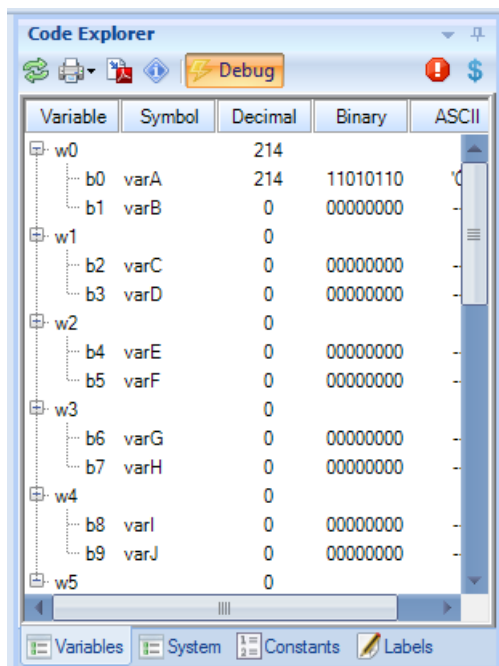
App – Variables appears under the simulation



The simulator window displays a memory map with addresses V, C.5, C.4, C.3, 08M2, C.0, C.1, and C.2. Below the memory map are playback controls (play, pause, next, stop) and a progress slider. At the bottom, a table shows the value of variable 'varA' as 0.

Variable	Value
varA	0

PE6 – Variables are shown in the Code Explorer



The Code Explorer window shows a list of variables and their values. The variables are organized into groups: w0, w1, w2, w3, w4, and w5. Each group contains several variables (b0-b9) with their respective decimal and binary values.

Variable	Symbol	Decimal	Binary	ASCII
w0		214		
b0	varA	214	11010110	
b1	varB	0	00000000	
w1		0		
b2	varC	0	00000000	
b3	varD	0	00000000	
w2		0		
b4	varE	0	00000000	
b5	varF	0	00000000	
w3		0		
b6	varG	0	00000000	
b7	varH	0	00000000	
w4		0		
b8	varI	0	00000000	
b9	varJ	0	00000000	
w5		0		

## Downloading a program into a PICAXE chip

### PE6 and App (not Cloud)

1. Connect your PICAXE project to the computer by the AXE027 USB download cable.
2. Connect power to the PICAXE circuit board, normally 3 x AA batteries (4.5V).
3. Note; your PICAXE chip, if already programmed, may start running the program from its memory – this will not affect the programming process.
4. Click the Program button on the PICAXE toolbar or press <F5>.
5. The programming progress window will appear.
6. Programming times vary depending on the type of chip and amount of program code – the larger the program, the longer the programming time.
7. If successful, programming is complete when the progress bar disappears.

If you are having difficulty programming try the hard reset procedure as described in part 1 of the PICAXE manual.

### Cloud (not PE6 or App)

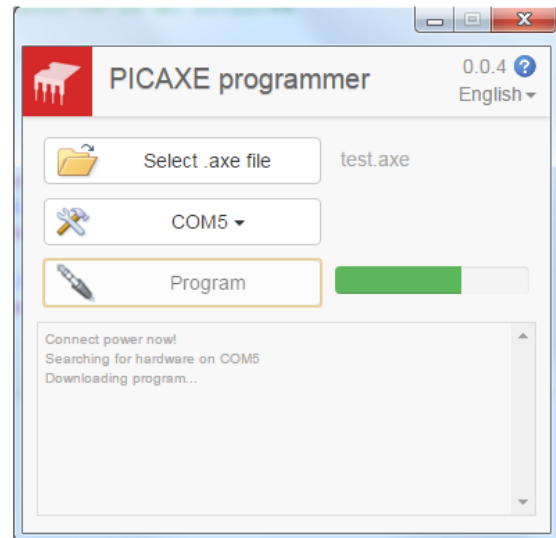
Web browsers do not allow web sites to access the USB port of your computer. This is a very sensible security restriction.

Therefore the online Cloud version of Blockly cannot program your chip directly (in the same way the app and PE6 versions can).

Therefore the Cloud version instead saves onto your computer an '.axe file' which is a compiled version of your PICAXE program.

You must then use the Chrome Programmer App ([www.picaxe.com/progapp](http://www.picaxe.com/progapp)) to download the .axe file onto the PICAXE chip.

## Using the Cloud Programmer App



1. Connect your PICAXE project to the computer by the AXE027 USB download cable.
2. Connect power to the PICAXE circuit board, normally 3 x AA batteries (4.5V).
3. Note; your PICAXE chip, if already programmed, may start running the program from its memory – this will not affect the programming process.
4. Open the desired .axe file and select the correct COM port.
5. Click the Program button
6. The programming progress window will appear.
7. Programming times vary depending on the type of chip and amount of program code – the larger the program, the longer the programming time.
8. If successful, programming is complete when the progress bar disappears.

If you are having difficulty programming try the hard reset procedure as described in part 1 of the PICAXE manual.

## Displaying and using BASIC

Blockly is also able to convert any complete program into BASIC or Javascript.

BASIC is a text based language that is used throughout the world to program everything from PICAXE microcontrollers to personal computers.

Javascript is a very common programming language used for developing web sites.

### Why Convert?

Although Blocks are easy to understand and quick to build. BASIC programming languages offer more complexity to advanced level users and the ability to convert a program into BASIC offers a way of learning how BASIC programs are written.

### Converting a program into BASIC

1. Design your program as normal and test the program using the program simulation tools.

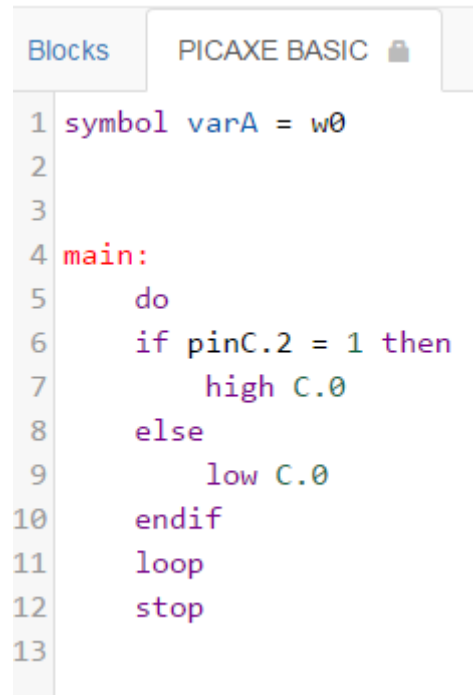
2. Convert to PICAXE BASIC

App - Click on the 'BASIC' tab

PE6 – Click on the 'Convert To BASIC' button

3 The BASIC text window is then displayed containing the conversion of your program.

*Note that it is also possible to display the BASIC tab in PE6, to do this use the File>Options>Diagnostics>Blockly>'Display BASIC' setting.*



The screenshot shows a software interface with a tab labeled 'PICAXE BASIC' and a lock icon. Below the tab is a text editor window with a line number column on the left (1-13) and BASIC code on the right. The code is: 1 symbol varA = w0, 2, 3, 4 main:, 5 do, 6 if pinC.2 = 1 then, 7 high C.0, 8 else, 9 low C.0, 10 endif, 11 loop, 12 stop, 13.

```
1 symbol varA = w0
2
3
4 main:
5 do
6   if pinC.2 = 1 then
7     high C.0
8   else
9     low C.0
10  endif
11  loop
12  stop
13
```

#### Notes:

Only blocks that are connected to Start block in your program are converted.

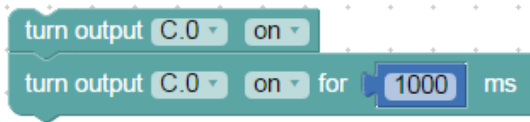
It is not possible to convert from BASIC backwards to blocks.

Using the BASIC block you can add sections of BASIC code into a program.

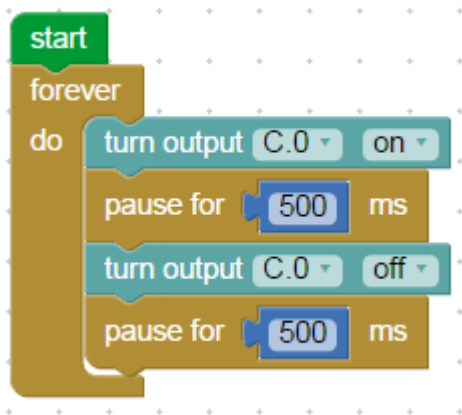
For full information on the use of BASIC to program PICAXE chips see the PICAXE website at [www.picaxe.com](http://www.picaxe.com)

## Section 2. Outputs

### Turn output on/off block



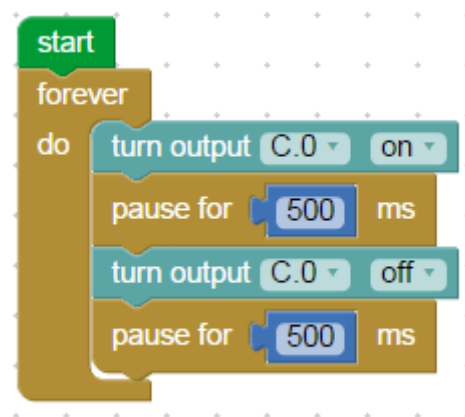
The turn output blocks are used to switch a single output on or off.



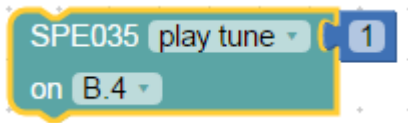
### Pause block



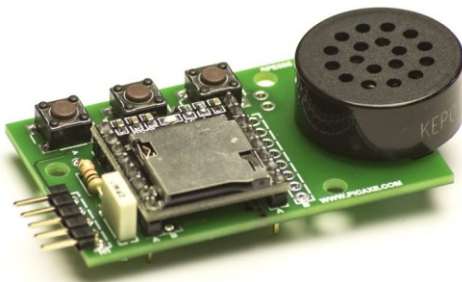
A pause block makes a running program pause for the number of milliseconds specified before the next block is carried out. You can use it to keep output devices switched on or off for a set time. Use its input box to enter a number of milliseconds or a Variable.



## SPE035 MP3 Player Block

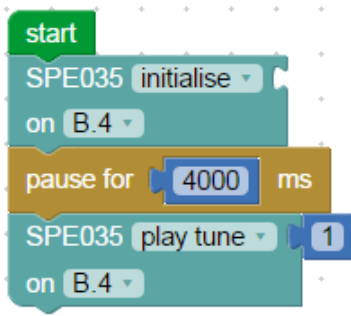


The SPE035 MP3 Player is a low cost device that can be used to add music and speech into your PICAXE project.



[www.picaxe.com/products/spe035](http://www.picaxe.com/products/spe035)

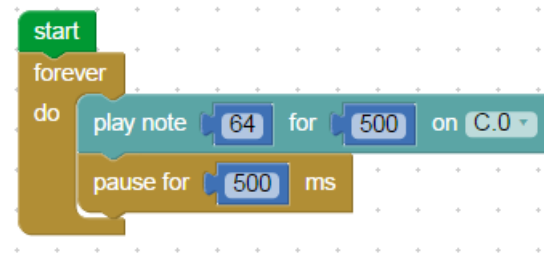
This block allows simple control of the SPE035 module to easily play back MP3 tunes.



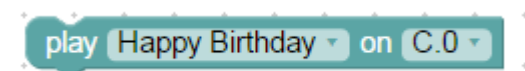
## Play Note Block



Use a play note block to send a pulsed signal to a piezo sounder connected to an output of a PICAXE microcontroller. You can use a sequence of sound blocks to play a simple tune.



## Play block



Most PICAXE chips have 4 pre-programmed internal tunes, which can be output via the Play block.

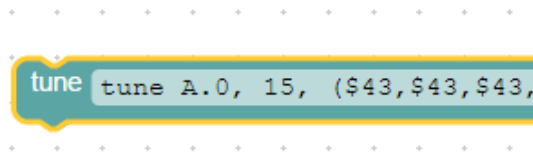
As these tunes are often included within the PICAXE bootstrap code, they use very little program memory.

The Tunes are:

- 0 - Happy Birthday
- 1 - Jingle Bells
- 2 - Silent Night
- 3 - Rudolf the Red Nosed Reindeer



## Tune block



Working in a similar way to the Play block, the Tune block allows special musical tunes to be played.

The difference with Tune block is that it converts RTTTL mobile phone ringtone files to PICAXE tunes and plays them with or without flashing outputs.

RTTTL ringtone files are freely available on the internet (there is a very wide range of tunes available) and these can be downloaded as small text files. The files contain the notes and timings that make up the tune. The Tune Wizard converts these ringtones to a PICAXE tune block upon download.

Once you have downloaded your ringtone file (ensure it is an RTTTL format), save it to disk.

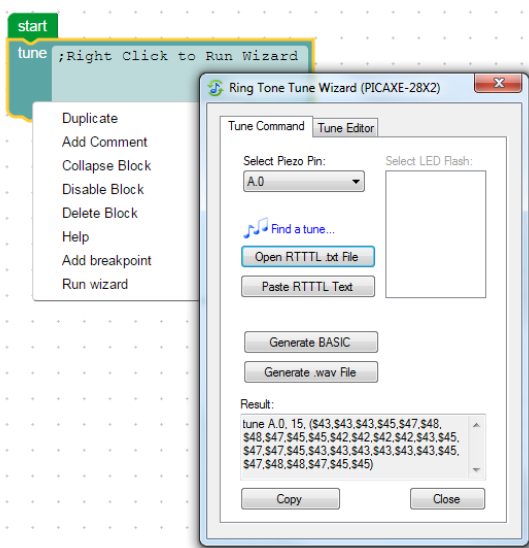
In PE6 you can right click the Tune command to start the Wizard. For the App you will need to run the Wizard separately.

Click the 'Open RTTTL txt file' button to browse the computer to find the file.

Select the output to flash using the drop down box. The chosen outputs switch on/off in time to the tune. The Flash Mode can switch outputs 0 and 4. Ensure that you have configured the I/O pin 4 as an output using the Select PICAXE dialog in order to see all of the available options.

Once you have generated the Tune BASIC click 'Copy' in the Wizard so that you can paste to code back into the Tune block.

Note that, unlike the Play Block, the Tune block requires much more memory in the chip as all of the notes have to be specially programmed into the chip. If you wish to play your tune a number of times, use the Tune block in a Procedure to save memory.



## Servo Block



Servos, as commonly found in radio control toys, are a very accurate motor/gearbox assembly that can be repeatedly moved to the same position due to their internal position sensor. Generally servos require a pulse of 0.75 to 2.25ms every 20ms, and this pulse must be constantly repeated every 20ms. Once the pulse is lost the servo will lose its position.

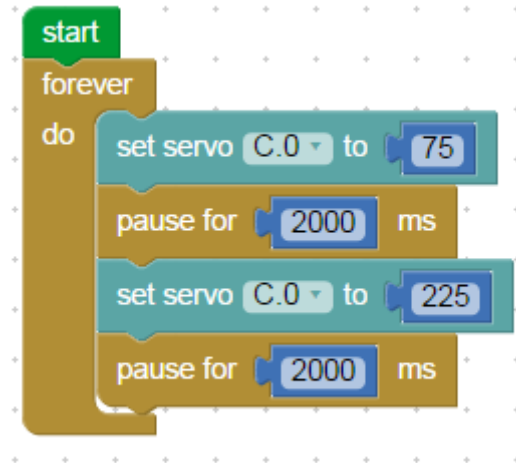
The Servo block starts a pin pulsing high for length of time pulse (x0.01 ms) every 20ms. This block is different to all other blocks in that the pulsing mode continues until another servo block or outputs block. Outputs blocks stop the pulsing immediately. Servo blocks adjust the pulse length to the new pulse value, hence moving the servo.

The block details for the servo block have two settings; the output pin that the servo motor is connected to and the pulse time.

The pulse time can be a value held in a Variable. Note that the value for the pulse time MUST be in the range 75 to 225. The servo motor may malfunction if the pulse is outside of this range.

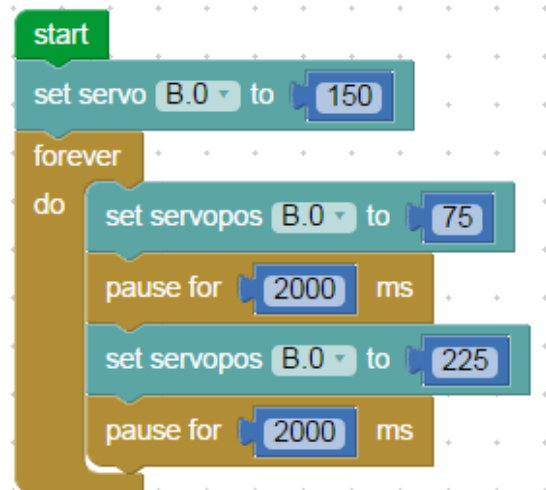
### Example

The program below will move a servo motor attached to output B.2 from one extent of its travel to the other, repeating continually.



*Using the Servo block*

Note that the servo block is required to 'activate' the servo pulsing. Once pulsing is started smoother operation (less jitter) can be achieved by using the 'servopos' command for subsequent movements.



## Send Infrared block



This block is used to transmit the infrared data to a Sony™ protocol device. It can also be used to transmit data to another PICAXE circuit that is using the 'read infrared' block. Data is transmitted via an infrared LED (connected on output 0) using the SIRC (Sony Infrared Control) protocol.

The send infrared block can be used to transmit any of the valid TV data codes (0-127). Note that the Sony protocol only uses 7 bits for data, so data codes of value 128 to 255 are not valid.

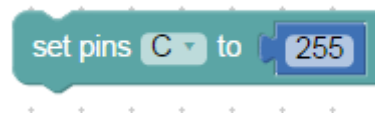
See also the 'read infrared' block.

## Pwmout block



The pwmout block is used to provide a varying ratio pulsed output. This is often used for speed control of motors. See the main PICAXE manual for more details.

## Set Pins block



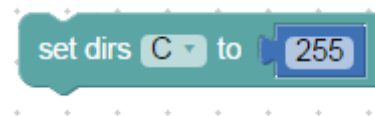
When program flow passes through a 'set pins' block, the output port is set to the binary value of the number entered in the block.

If you are familiar with the binary system then the set pins block is a convenient way of switching combinations of outputs on or off.

bit	7	6	5	4	3	2	1	0
value	128	64	32	16	8	4	2	1

In the table above the 'bits' can be switched on by sending the selection value of the bit., e.g. 'set pinsB to 4, which will turn on an LED at B.2 (and switch off all the other outputs.

## Set Dirs block

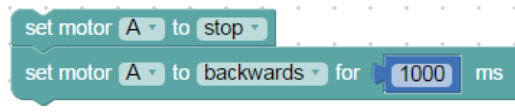


The set dirs. Block converts pins to either input or outputs. A binary bit value of 1 means output, a value of 0 means input.

Commands such as 'turn output on' will also automatically set the dirs bit to an output.



## Motor blocks



The Motor block allows you to use pairs of outputs on a PICAXE microcontroller to switch a motor forward, reverse or off.

The motors are given a letter A to D that controls outputs as follows:

	Non-8 pin	08M2
A	(B.0, B.1)	(C.0, C.1)
B	(B.2, B.3)	(C.2, C.4)
C	(B.4, B.5)	
D	(B.6, B.7)	

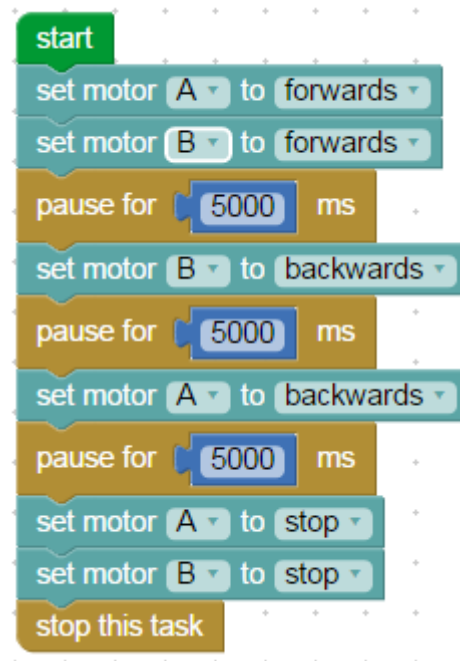
Remember that the direction in which a motor turns depends on which way current flows through it, and therefore on the way it is connected to power. Therefore if your motor moves in the wrong direction it may be necessary to reverse the two wires.

NOTE: Output and Motor blocks both use the same output pins to switch the outputs of a PICAXE microcontroller.

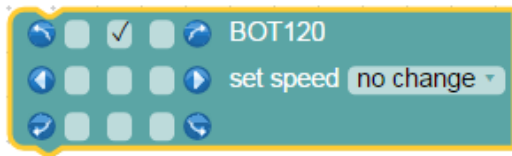


### Example

A steerable buggy is usually driven by two motors, one powering each driving wheel with a free-running jockey wheel to keep it stable. The program below shows how a sequence of Motor blocks can be used to drive a buggy which has one motor connected to outputs 0 and 1 (motor A) and the other motor connected to outputs 2 and 3 (motor B).



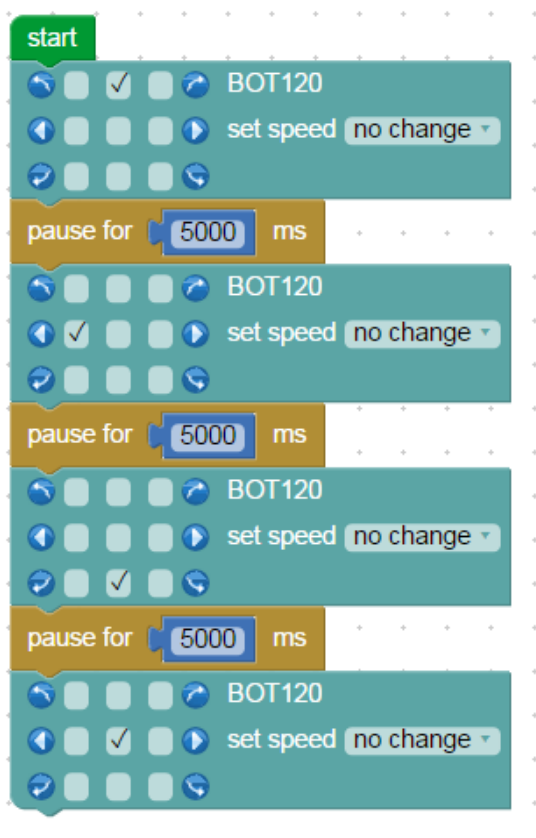
## Robot Motor blocks (various)



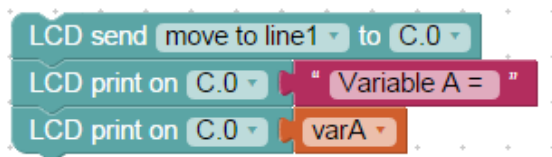
Some robots such as the BOT120 Microbot use 'pairs' of motors (Motors C and D) to control their movement.

For these robots it is easier to use the dedicated robot motor blocks, as these blocks will automatically control both of the robot's motors.

For instance to make the robot go forwards select the centre checkbox in the top row. This will switch on both motors C and D in the forward direction



## LCD blocks



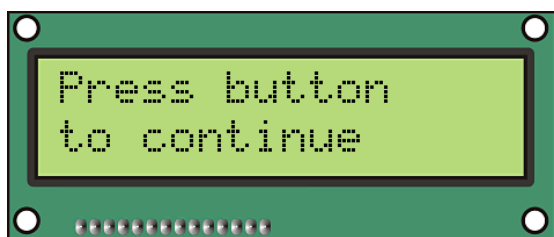
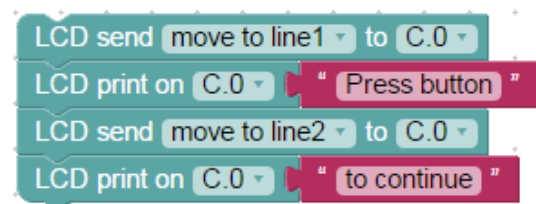
These block can be used to display a message on an LCD screen attached to a PICAXE-driven circuit board.



[www.picaxe.com/products/axe133y](http://www.picaxe.com/products/axe133y)

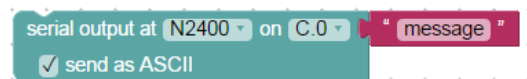
This block will be simulated if the program is simulated within PE6 (not within app).

Within PE6 a small LCD screen window will pop up during the run to display the LCD message (make sure the simulation pin for the LCD is correctly set under File>Options>Simulation)



Simulated LCD screen

## Serout Block



This block allows output information to be sent from the PICAXE microcontroller to a device such as a serial printer, a serial LCD screen or another PICAXE which is connected to an output of a PICAXE microcontroller.

The first list to set is the serial mode. Set the mode to that specified by the device you are sending data to.

The second list is used to select the output pin on the PICAXE microcontroller to send the data through.



The data to be sent is attached to the input on the right. This can be a text string, a variable, or a constant.

If using a string make sure you also check 'send as ASCII'.

If using a variable or constant you can either send the raw binary value e.g 32 (no ASCII check) or the ASCII character equivalents e.g. "3" then "2"(check ASCII box)

## Sertxd Block



The sertxd block is similar to the serout block, but acts via the serial output pin rather than a general output pin. This allows data to be sent back to the computer via the programming cable. This can be useful whilst debugging.

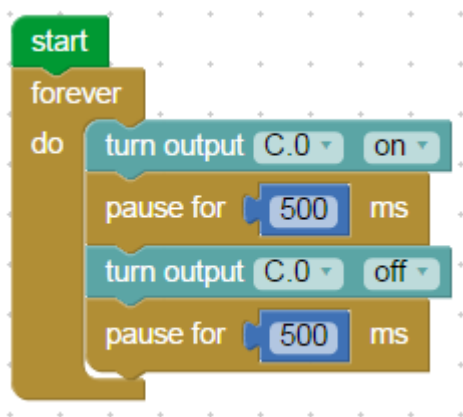
See the PICAXE Manual 2 for more information

## Section 3. Delays

### Pause block



A pause block makes a running program pause for the number of milliseconds specified before the next block is carried out. You can use it to keep output devices switched on or off for a set time. Use its input box to enter a number of milliseconds or a variable.

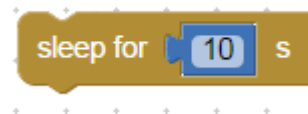


### Wait until Input block



This block waits until an input pin changes state – until the input goes on (high) or off (low).

### Sleep block



This block puts the PICAXE microcontroller into low power mode for a specified number of seconds.

This block can be used to save battery power in your project. All output devices will be left in their current condition, but signals from input devices will not be responded to while the chip is in sleep mode.

The time input box is used to set the number of seconds of sleep mode required.

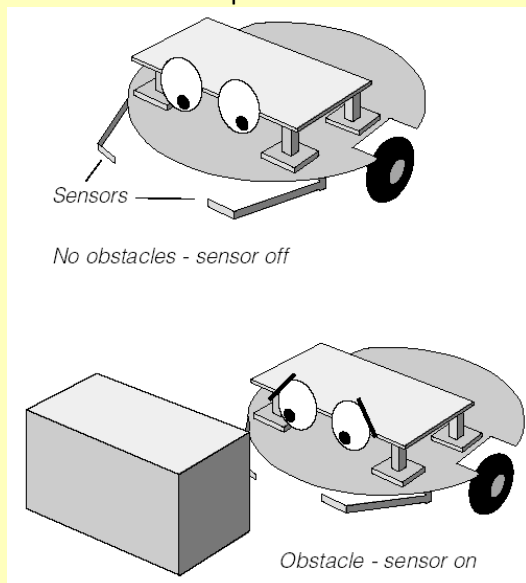
Note that Sleep times are not as accurate as Pause times.

## Section 4. Inputs

Input devices such as switches and sensors send information from the outside world into the control system. Output devices are switched on or off in response to the information provided by input devices.

### Example

A buggy is often fitted with micro-switches so that if it approaches an obstacle, a microswitch will be pressed.



The information that the switch has been pressed can be used in the system to switch off the motors driving the buggy, and start a sequence of movements to move around the obstacle.

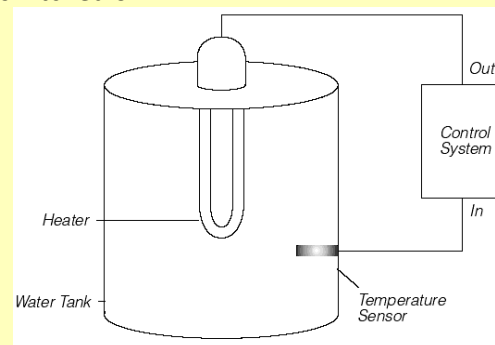
A microswitch is a digital sensor. It has only two states - "on" (or "closed") and "off" (or "open").

These states are often labelled by the digits 1 and 0, which is why the sensors are called digital sensors.

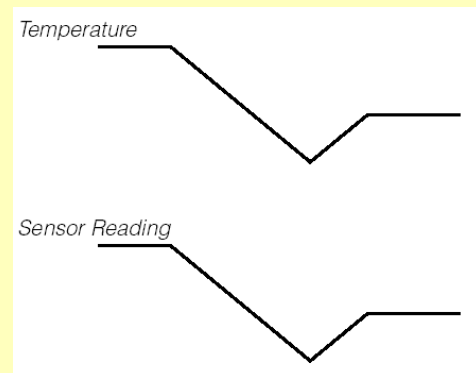
### Example

A controlled hot water system includes a temperature sensor which constantly monitors the water temperature.

The water heater is switched on and off in response to the information provided by the sensor. If the water temperature falls below a set level, the heater is switched on until it reaches that level again. Then the heater is switched off.



A temperature sensor is an analogue sensor. It provides a reading which changes in line with the changing level of whatever it is sensing.



## Input block

```
if input C.1 is on
then
  turn output C.0 on
else
  turn output C.0 off
```

Use this block to test the state of a digital sensor connected to a digital input of a PICAXE microcontroller.

When program flow reaches the input block it will only process one of the two sections, dependant on whether the input pin is on (high) or off (low).

Multiple input blocks can also be nested to test, for instance, if two pins are both on.

```
if input C.1 is on
then
  if input C.2 is on
  then
    turn output C.0 on
  else
    turn output C.0 off
else
  turn output C.0 off
```

## Analogue Block

### Temperature Block

### Ultrasonic Block

```
read analogue C.1 to varA
read temperature C.1 to varA
read ultrasonic C.1 to varA
```

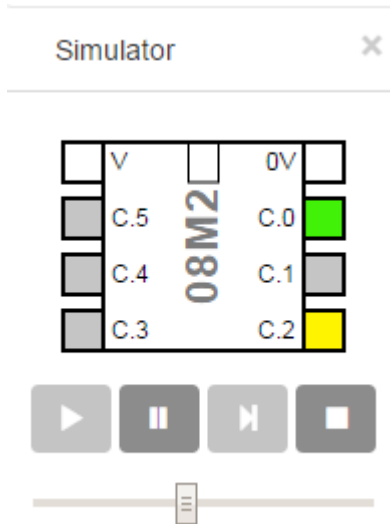
These three blocks all operate in the same way, loading an analogue value into the select variable. To then test whether the desired threshold point has been reached a variable decision block is then used.

Analogue – any generic sensor like an LDR  
Temperature – DS18B20 temperature sensor  
Ultrasonic – SRF005 distance sensor

```
start
forever
do
  read analogue C.1 to varA
  if varA > 100
  then
    turn output C.0 on
  else
    turn output C.0 off
```

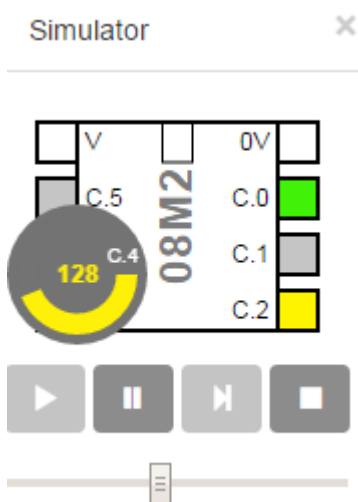
## Simulating a Digital Input

To change the state of an input simply click on the input pin within the simulation. It will turn from grey (off) to yellow (on).



## Simulating an Analogue Value

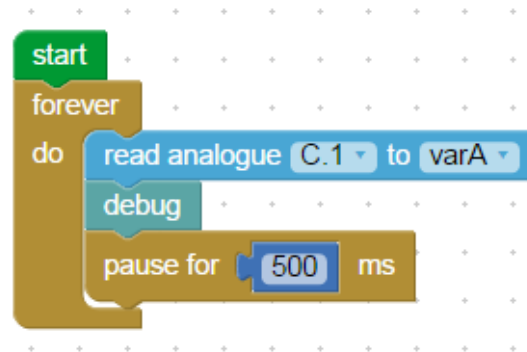
To simulate a changing value for analogue blocks right click over the corresponding pin in the simulation panel and use the radial slider.



## Calibrating a Sensor using Debug

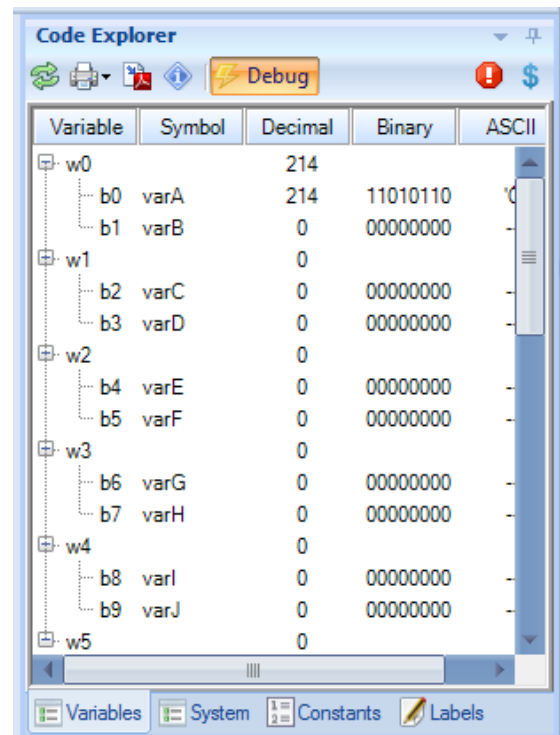
Often when using analogue sensors it is necessary to experiment to find the correct threshold point (e.g. the correct light value to switch a device on or off).

To read analogue values 'live' from a PICAXE chip we can also use the Debug block in a loop like in the program below.

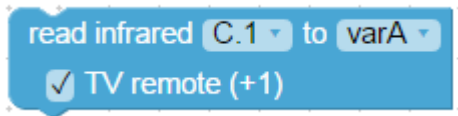


Click on 'Debug' (in PICAXE menu in app or on the Code Explorer panel in PE6).

The value of varA then be displayed on the computer screen and will update every 500 milliseconds.



## Read Infrared Block

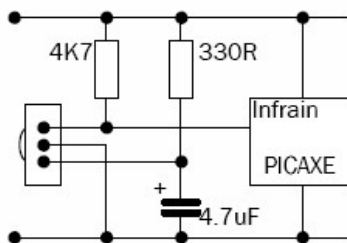


To receive information from an Infrared source, the Read Infrared block is used. The block will wait for a new infrared signal from an infrared TV style transmitter. It can also be used to receive a 'send infrared' block sent out from a separate PICAXE chip.

All processing stops until the new block is received. The value of the code received is placed in the chosen variable.

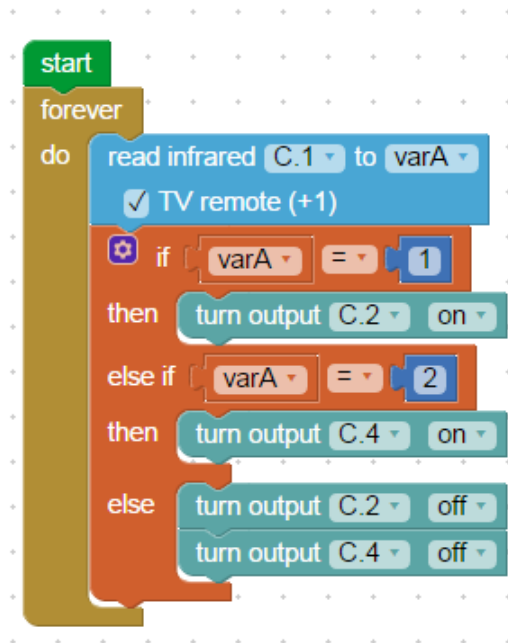
As TV style remotes (e.g. part TVR010) send out an offset value (e.g. value 0 for channel 1, value 1 for channel 2) the checkbox allows you to automatically add 1 to the value received, so the variable value then matches the button pressed.

The basic circuit required for this block is as follows. The device on the left side of the circuit is an IR receiver LED, part code LED020.



### Example

In the following program a signal is received from a TV Infrared remote control. Lights are switched on if key 1 or key 2 is pressed.

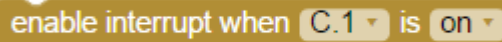


The read infrared block waits until a signal is received, and saves this as a number in Variable A.

If the value is 1 or 2 the outputs are switched on. Otherwise the lights are switched off.



## Interrupt Block

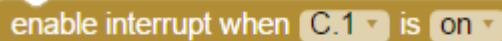


enable interrupt when C.1 is on

An Interrupt instantly captures the flow of control whenever a preset digital input condition occurs to trigger it e.g. when a switch is pressed.

When the interrupt is triggered flow jumps immediately to a sub-procedure block (which must be called 'interrupt') and then carries out any blocks which follow until it reaches the end of that block. It then returns to the point in the main program which it was at when the Interrupt occurred.

In order to use an Interrupt, the PICAXE must be told to look for the input condition. This is done through the enable interrupt block.

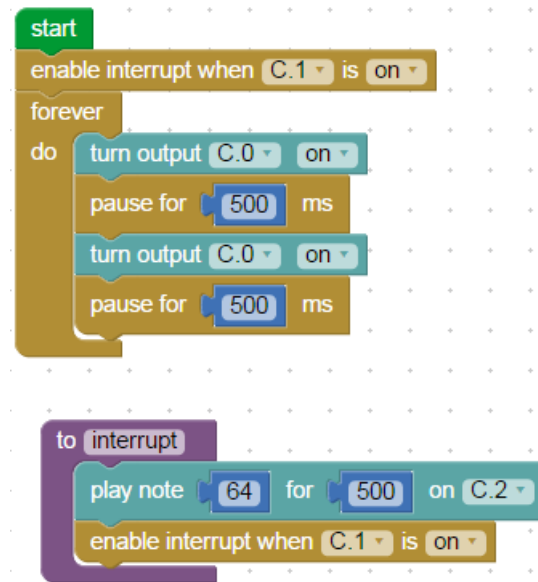


enable interrupt when C.1 is on

To prevent the Interrupt retriggering itself, the Interrupt is automatically disabled once it is triggered. To re-enable it another enable interrupt block is required.

### Example

A PICAXE microcontroller running a continuous loop flashing lights needs to be able to react to a button press and play a warning sound.

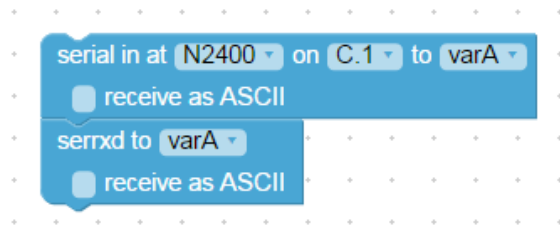


The Interrupt is used to capture the flow and play a sound. The interrupt is then enabled once again before returning to the point at which it left the main flow.

There is no limit to the number of blocks inside the Interrupt. It is a common technique to add an 'Enable Interrupt' block just before the end of the procedure, so that when the Interrupt sub procedure returns the interrupt is re-enabled.

Only one Interrupt sub-procedure can be used per program.

## Serial In Block



The Serial In block is used to receive serial data into an input pin of the microcontroller. It cannot be used with the serial download input pin, for this pin use the serrxd command instead.

The input pin is the input on the PICAXE that the data is to be received through. The Variable option is a variable location that the data is stored into once it is received.

The first option specifies the baud rate and polarity of the signal. When using simple resistor interface, use N (inverted) signals. When using a MAX232 type interface use T (true) signals. The protocol is fixed at N,8,1 (no parity, 8 data bits, 1 stop bit).

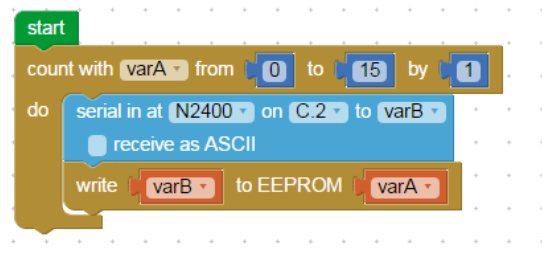
The Serial In block forces the PICAXE chip to wait until serial data is received through the chosen input. This data is stored in the chosen variable.

To store raw binary data (as opposed to ASCII strings) make sure the ASCII checkbox is not checked.

### Example

Serial data is being received from another PICAXE chip and needs to be stored in the EEPROM.

In the program shown below, the serial data is read into Variable B through input pin C.2. The Write block is used to store the value in Variable B in the EEPROM. This process is repeated 16 times to fill the EEPROM memory locations

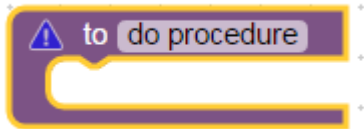


*Using the Serial In block to receive serial data*

## Section 5. Procedures

Blockly provides a clear, step-by-step method of building a complex control system, by creating a number of linked subsystems called “sub procedures”.

### How to build a sub procedure

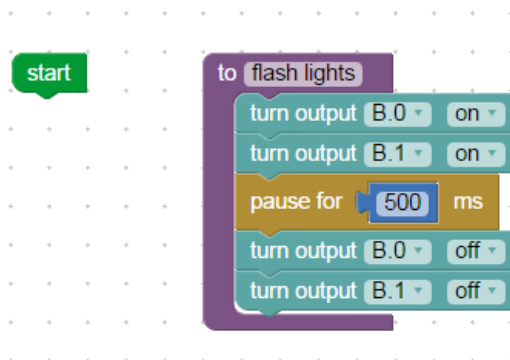


Drag out a ‘to do Procedure’ block to begin the procedure. Drag the block onto the program and place it separately from the start block as shown below.

Change the name to something related to the task it does.

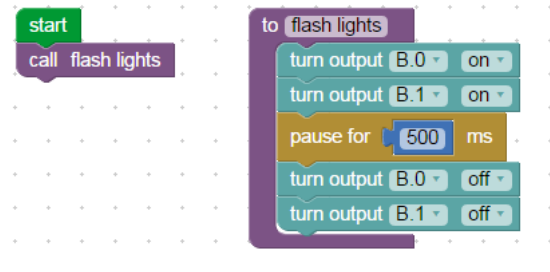


Use other blocks as normal to create the procedure.



### How to use a procedure

Once you have built a procedure, you can call it into use whenever you like in the program by using the call block, as shown below.



The call block calls the procedure into use.

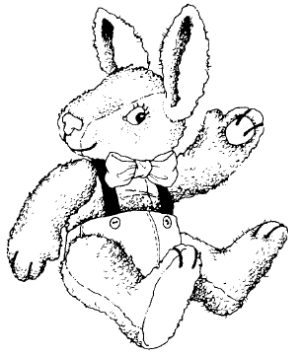


Drag the call block onto the program. Place it at the point where you want the procedure to be called into use.

Note that all the procedures that have been built in a program are automatically listed in the toolbox. When flow reaches a call block, it jumps to the Procedure block with the same name. When the program flow reaches the end of the procedure, the flow jumps back to the call block that called the procedure. To test run the whole program click on the simulation Run button.

### Example

A PICAXE microcontroller is used to control a system in a child's toy which plays a tune when it is hugged. A piezo transducer is connected to an output pin, and a push switch is used to sense when the toy is hugged. The program for the system is shown below. The tune is created as a procedure which can be tested and edited separately from the main routine.

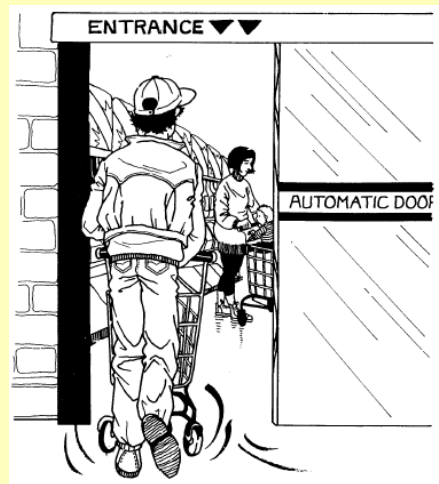


```
start
forever
do
  if input C.0 is on
  then call play_tune

to play_tune
  play Happy Birthday on B.0
```

Using a Procedure to play a tune after an input condition is met

### Example



The program shown below is a control system for a sliding door. When a switch is pressed, the door opens. It stays open for ten seconds and then closes again.

```
start
forever
do
  if input C.0 is on
  then
    call open_door
    pause for 10000 ms
    call close_door

to open_door
  set motor A to forwards
  pause for 500 ms
  set motor A to stop

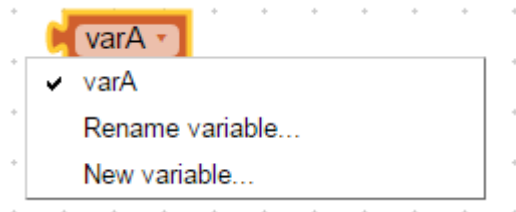
to close_door
  set motor A to backwards
  pause for 500 ms
  set motor A to stop
```

Sliding door control system using procedures

## Section 6. Maths & Variables

In Blockly a variable is a 'number container' that can hold a given value between 0-65535.

The variables are called varA to varZ by default, but can be renamed to any name you choose. To rename a variable simply click on the drop down arrow to the right of the name and select 'Rename Variable...'



This section explains how they can be used for a variety of mainly counting and timing purposes.

The current value of a variable can be seen during a simulation in the Code Explorer panel (PE6) or under the simulation panel (app).

Simulator ×

---

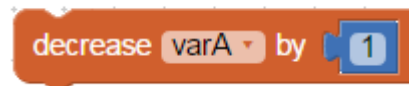
Variable	Value
varA	0

### Increase Variable block



Each time flow passes through an Increase block, the desired value is added to the value of the selected variable.

### Decrease Variable block



The decrease block works in a very similar way to the increase block.

The difference is that when program flow passes through a decrease block, the value is subtracted from the selected variable.

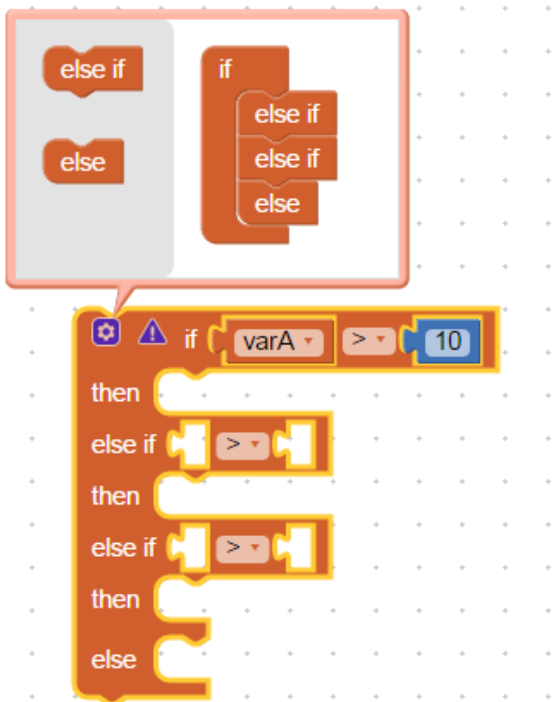
Note that PICAXE chips do not support negative numbers, so any number below 0 will underflow e.g. -1 actually becomes 65535.

## Variable If block



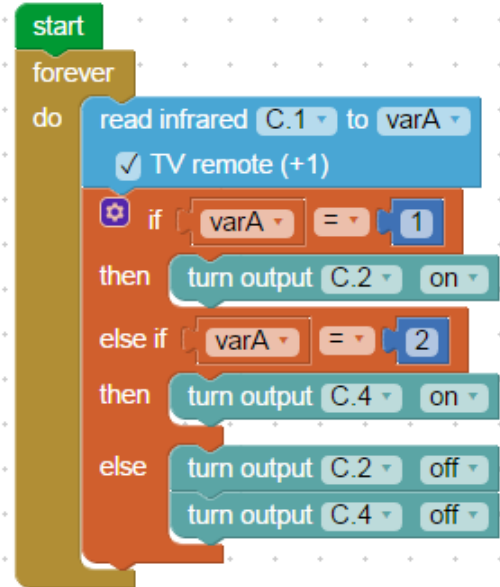
To test the variable the 'variable if' block is used. The value can be tested to see if it is greater, less than or equal to another variable or a set value.

This block is unique in that its shape can be modified to add more tests. To do this click the blue 'settings' icon top left and then, working completely within the small pop-up window, configure the block by dragging out the number of else if commands required.



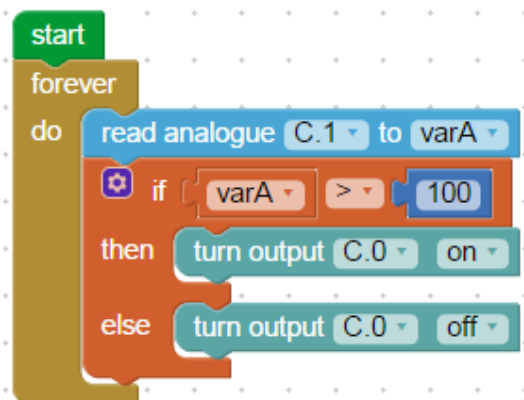
### Example one

Using this system you can configure the blocks for a number of consecutive tests, as shown in this infrared example.



### Example two

A PICAXE microcontroller is being used to control a lamp. A light sensor is connected to analogue input 0. The system will switch on the lamp automatically in dark conditions. Below is a program for the system.



*Example three*

A PICAXE microcontroller is used to control a system for counting cars entering and leaving a car park using two digital sensors. When 10 cars are in the car park the 'FULL' light is lit.



```
start
set varA to 0
forever
do
  if input C.0 is on
  then increase varA by 1
  if input C.1 is on
  then decrease varA by 1
  if varA > 10
  then turn output B.0 on
  else turn output B.0 off
```

*Program for making and displaying a count.*

## Set Variable block



A value can also be given to a variable in the form of a mathematical expression as shown in the example above ( $\text{varA} = \text{varC} + \text{varB}$ ).

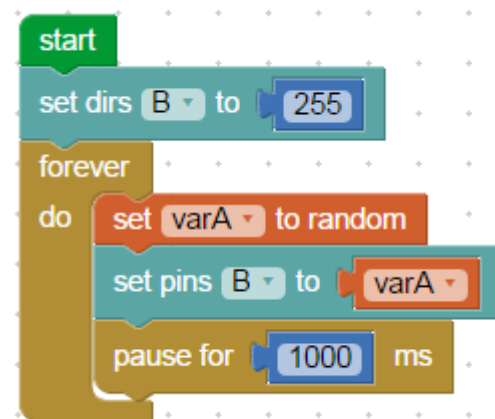
Note how multiple expressions can also be nested inside each other to make the equation longer ( $\text{varA} = \text{varC} + \text{varB} + 10$ ).



## Random block



Using the random block a variable can be given a random value between 0 and 65535. In the example shown below, a set of display lights for a small Christmas tree are connected to 8 outputs of a PICAXE microcontroller. Every second the display will change at random.

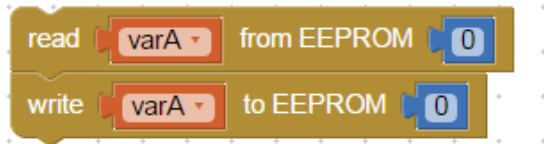


Note that as with all microcontrollers and computers, the generation of random numbers is based on a sequence.



## Section 7. Advanced Blocks

### Read and Write Blocks

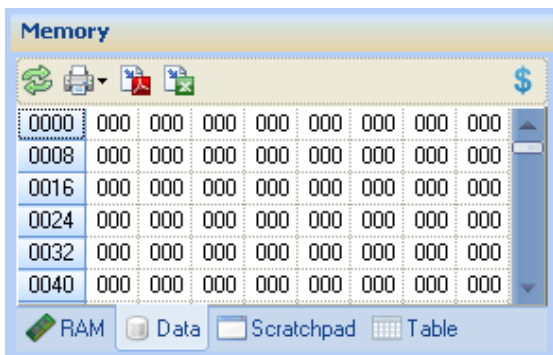


When a program run is started, all variable values automatically reset to zero. So, when the PICAXE microcontroller is reset or powered up, all variable values are reset to zero.

If you want to retain variable values when the PICAXE microcontroller is powered up or reset, you can use the write block to store values in the chip's data EEPROM memory. The read block is used to retrieve the values from the chip's memory.

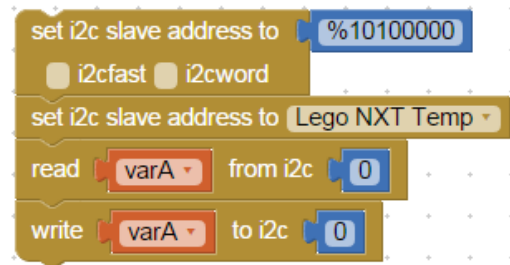
The read block takes the value which is currently stored in a selected address (in this case address 0), and puts it into the selected variable (in this case variable A).

The PICAXE microcontroller's data EEPROM memory has 255 separate addresses. Each address can store a number between 0 and 255. The EEPROM window (PE6 only) displays the contents of the memory when you test run a program.



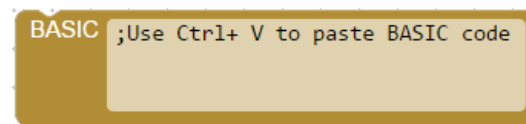
Data EEPROM window

### Read I2C and Write I2C Blocks



The I2C commands allow communication with third party devices such as memory EEPROMs and sensors. For further details see the main PICAXE manuals.

### BASIC



This block is used as an extension to a program. Any valid PICAXE BASIC code can be typed into the block cell window. When program flow arrives at this block the BASIC code within the block will be processed as if it were a procedure.

Note that only PE6 can simulate the BASIC code, the app/cloud versions of Blockly will skip over this block. This is because PE6 contains a much more powerful simulator engine than the web versions.

*For information on the other advanced commands (e.g. suspend, reconnect etc.) please see the corresponding description in PICAXE manual 2 (BASIC commands).*

## Section 8. Licenses

'Blockly for PICAXE' makes use of the following open source projects and acknowledges the developers of these projects.

<a href="https://github.com/google/blockly/blob/master/COPYING">Blockly</a>	<a href="https://github.com/google/blockly/blob/master/COPYING">https://github.com/google/blockly/blob/master/COPYING</a>
<a href="https://github.com/twbs/bootstrap/blob/master/LICENSE">Bootstrap</a>	<a href="https://github.com/twbs/bootstrap/blob/master/LICENSE">https://github.com/twbs/bootstrap/blob/master/LICENSE</a>
<a href="https://github.com/eligrey/FileSaver.js/blob/master/LICENSE.md">FileSaver</a>	<a href="https://github.com/eligrey/FileSaver.js/blob/master/LICENSE.md">https://github.com/eligrey/FileSaver.js/blob/master/LICENSE.md</a>
<a href="https://github.com/eligrey/Blob.js/blob/master/LICENSE.md">Blob</a>	<a href="https://github.com/eligrey/Blob.js/blob/master/LICENSE.md">https://github.com/eligrey/Blob.js/blob/master/LICENSE.md</a>
<a href="https://github.com/codemirror/CodeMirror/blob/master/LICENSE">CodeMirror</a>	<a href="https://github.com/codemirror/CodeMirror/blob/master/LICENSE">https://github.com/codemirror/CodeMirror/blob/master/LICENSE</a>
<a href="https://github.com/NeilFraser/JS-Interpreter/blob/master/LICENSE">JS-Interpreter</a>	<a href="https://github.com/NeilFraser/JS-Interpreter/blob/master/LICENSE">https://github.com/NeilFraser/JS-Interpreter/blob/master/LICENSE</a>
<a href="https://github.com/ternjs/acorn/blob/master/LICENSE">Acorn</a>	<a href="https://github.com/ternjs/acorn/blob/master/LICENSE">https://github.com/ternjs/acorn/blob/master/LICENSE</a>
<a href="https://github.com/jquery/jquery/blob/master/LICENSE.txt">jQuery</a>	<a href="https://github.com/jquery/jquery/blob/master/LICENSE.txt">https://github.com/jquery/jquery/blob/master/LICENSE.txt</a>
<a href="https://github.com/janl/mustache.js/blob/master/LICENSE">Mustache.js</a>	<a href="https://github.com/janl/mustache.js/blob/master/LICENSE">https://github.com/janl/mustache.js/blob/master/LICENSE</a>
<a href="https://github.com/jrburke/requirejs/blob/master/LICENSE">RequireJS</a>	<a href="https://github.com/jrburke/requirejs/blob/master/LICENSE">https://github.com/jrburke/requirejs/blob/master/LICENSE</a>
<a href="https://github.com/millermedeiros/js-signals">Signals</a>	<a href="https://github.com/millermedeiros/js-signals">https://github.com/millermedeiros/js-signals</a>