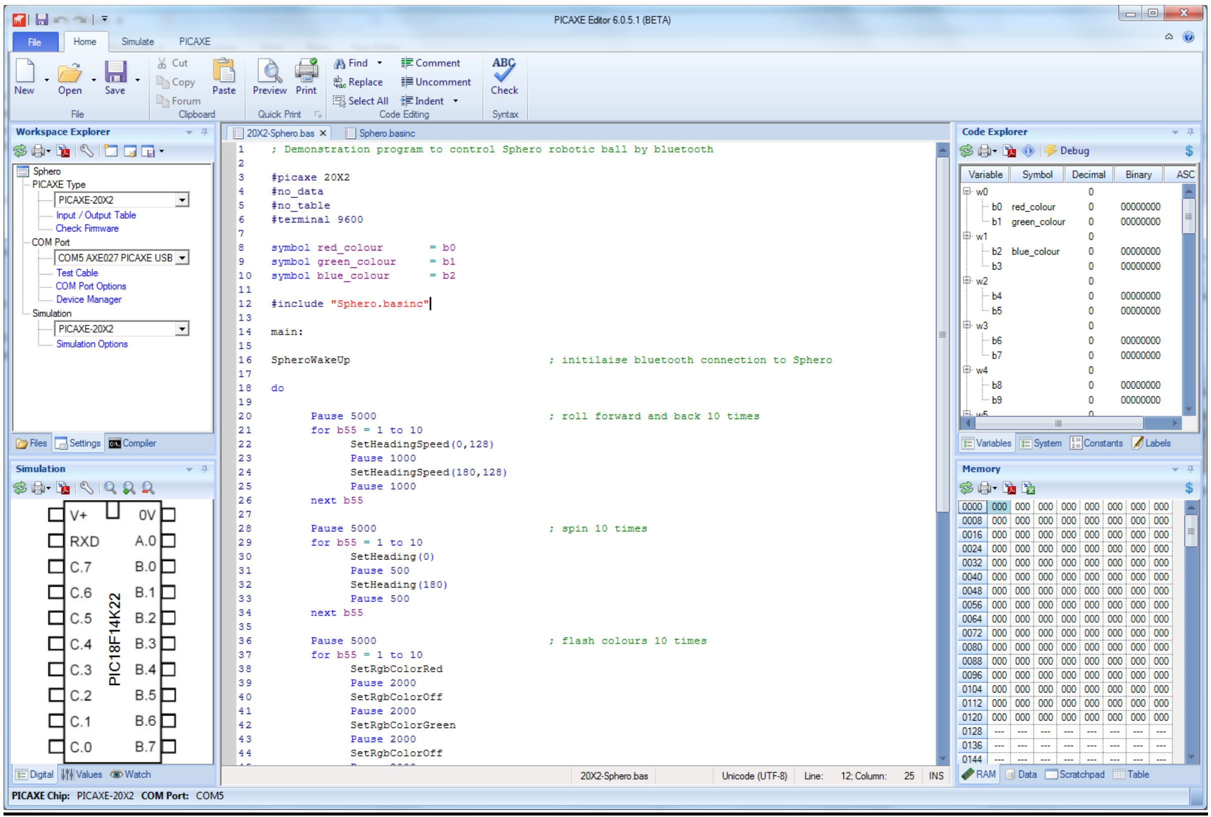
PE6 BETA TESTING BRIEFING

Thank you for helping test the beta version of the new PE6 software.
This document briefly highlights the main new features of the software.
Please send any feedback to support@picaxe.com using the subject 'PE6'

*We hope you don't ever see the beta software crash, but if you do please **do submit** the automated email report to us. Without this report we cannot rectify the issue!*



```
; Demonstration program to control Sphero robotic ball by bluetooth

#picaxe 20X2
#no_data
#no_table
#terminal 9600

symbol red_colour       = b0
symbol green_colour     = b1
symbol blue_colour      = b2

#include "Sphero.basinc"

main:

SpheroWakeUp                               ; initilaise bluetooth connection to Sphero

do

      Pause 5000                           ; roll forward and back 10 times
      for b55 = 1 to 10
            SetHeadingSpeed(0,128)
            Pause 1000
            SetHeadingSpeed(180,128)
            Pause 1000
      next b55

      Pause 5000                           ; spin 10 times
      for b55 = 1 to 10
            SetHeading(0)
            Pause 500
            SetHeading(180)
            Pause 500
      next b55

      Pause 5000                           ; flash colours 10 times
      for b55 = 1 to 10
            SetRgbColorRed
            Pause 2000
            SetRgbColorOff
            Pause 2000
            SetRgbColorGreen
            Pause 2000
            SetRgbColorOff
```

Sphero
PICAXE Type: PICAXE-20X2
COM Port: COM5 AXE027 PICAXE USB
Simulation: PICAXE-20X2

20X2-Sphero.bas | Unicode (UTF-8) | Line: 12; Column: 25 | INS

PICAXE Chip: PICAXE-20X2 COM Port: COM5

Section 1: Background

The existing PICAXE Programming Editor (PE5) software is extremely popular and in daily use with hundreds of thousands of PICAXE customers. However with advances in computing PE5 has become quite dated and therefore needed a complete overhaul.

The very first PICAXE Programming Editor was little more than a very simple text editor with an added 'Program' and 'Debug' menu for programming the original PICAXE-28 chip, and was originally distributed on two floppy disks! Over the last 15 years the software has constantly evolved with many additional new features such as syntax colouring and on-screen simulations, as well as support for over 20 different types of PICAXE chip.

However the development platform used to create the PE5 application is now obsolete, which makes maintaining the current PE5 software both time consuming and feature limiting, with many commonly requested features such as interactive tooltips and embedded PDF support difficult to add.

PE6 (PICAXE Editor 6) is therefore a completely new application, built entirely from scratch with modern software development tools for a much more feature rich PICAXE Editor and simulator. None of the PE5 source code has been reused within PE6 – it really is entirely new (but the PICAXE compilers, the separate programs that convert your BASIC into code, are exactly the same).

PE6 also includes a powerful new pre-processor that adds support for #include files and macros.

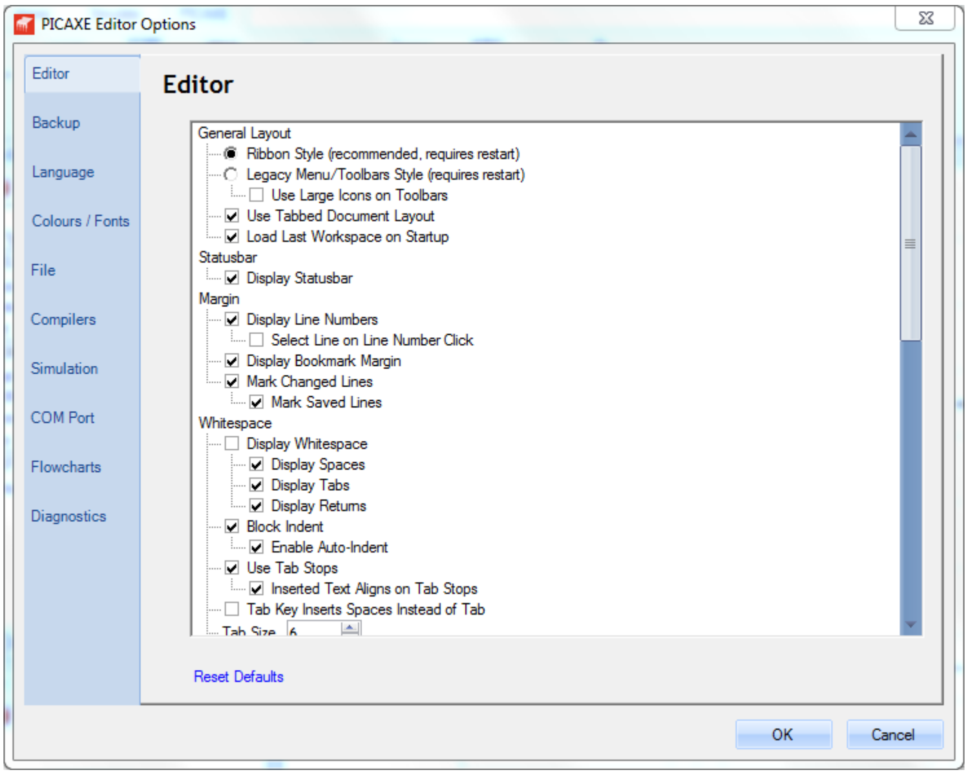
By moving to a modern development platform our software developers can also now work far more productively. Furthermore the entire PICAXE software architecture has simultaneously been moved over to a much more modular system, allowing future additions and new features (e.g. new PICAXE chip generations or extra code assistance wizards) to be added much more rapidly.

Finally thank you for your continued PICAXE support. Almost all of the new PE6 features are suggestions and requests from existing PICAXE users. We haven't quite managed the magic automated 'Fix my code' button yet, but we do have well over 200 other customer feature requests already working within PE6!

Change is always hard, it will naturally take you a short while to get used to all the new PE6 features and how project workspaces operate.
But once you've used it for a while, we think you will enjoy!

PE6 – the way you want it…

Remember almost all features can be enabled/disabled/customised via the File>Options dialog. Please do spend 5 minutes looking through all the Option tabs.



Editor

General Layout
- ● Ribbon Style (recommended, requires restart)
- ○ Legacy Menu/Toolbars Style (requires restart)
  - ☐ Use Large Icons on Toolbars
- ☑ Use Tabbed Document Layout
- ☑ Load Last Workspace on Startup

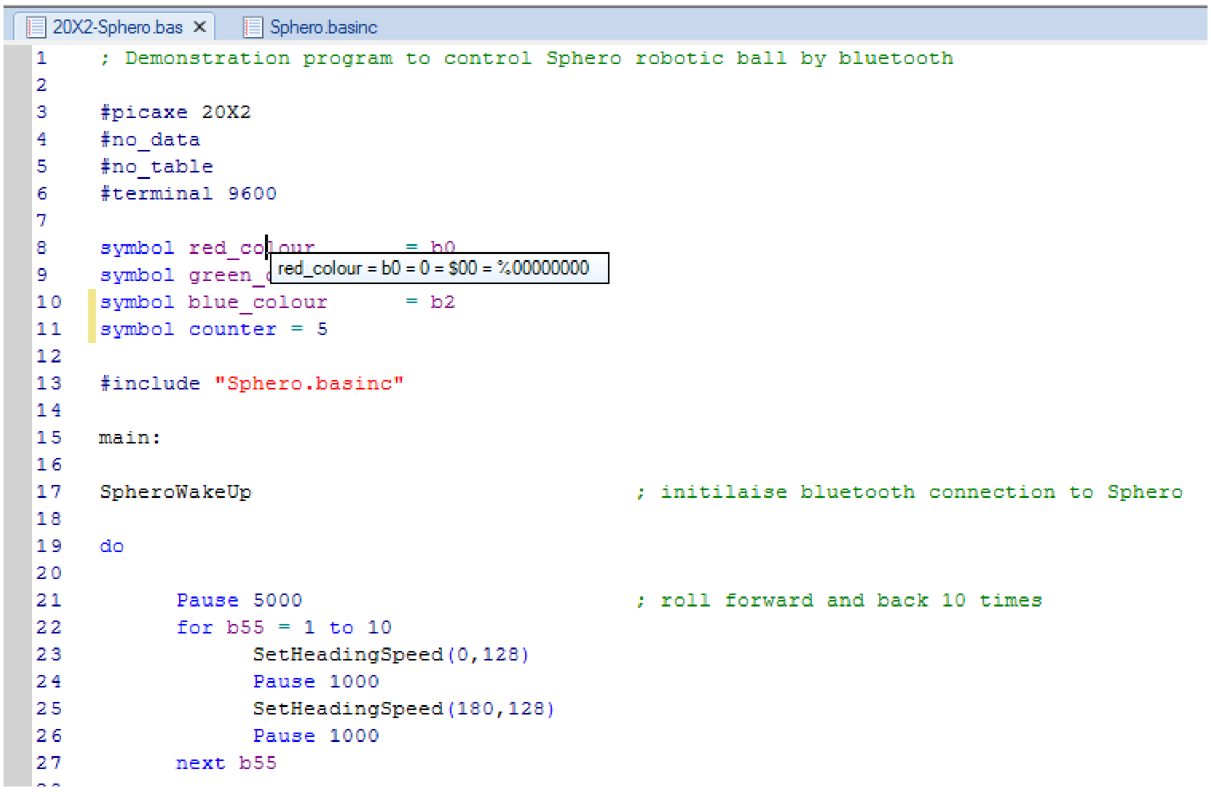
Statusbar
- ☑ Display Statusbar

Margin
- ☑ Display Line Numbers
  - ☐ Select Line on Line Number Click
- ☑ Display Bookmark Margin
- ☑ Mark Changed Lines
  - ☑ Mark Saved Lines

Whitespace
- ☐ Display Whitespace
  - ☑ Display Spaces
  - ☑ Display Tabs
  - ☑ Display Returns
- ☑ Block Indent
  - ☑ Enable Auto-Indent
- ☑ Use Tab Stops
  - ☑ Inserted Text Aligns on Tab Stops
- ☐ Tab Key Inserts Spaces Instead of Tab
- Tab Size 6

Key New Feature Summary:

- New modern ribbon interface
- Project workspace saves multiple files and individual project settings
- Display of images and diagrams as well as code files
- New pre-processor allows code to be split between multiple files and new macro feature
- Tabbed document interface
- Interactive tooltips over BASIC code
- Interactive outlining e.g. do..loop, if..endif
- Improved printing with print preview and embedded PDF export
- Automated file backup
- Multiple file type export
- Improved and extended editor features such as find/replace and block comment.
- Improved scrollbars and intellimouse support.
- Debug sessions background integrated into Code Explorer panel
- Improved simulation engine
- Improved Serial Terminal
- New improved flowcharting mode
- Remote download feature for compiled programs (Windows/Mac/Linux/Android)
- Unicode text support



```
; Demonstration program to control Sphero robotic ball by bluetooth

#picaxe 20X2
#no_data
#no_table
#terminal 9600

symbol red_colour          = b0
symbol green_
symbol blue_colour         = b2
symbol counter = 5

#include "Sphero.basinc"

main:

SpheroWakeUp                           ; initilaise bluetooth connection to Sphero

do

      Pause 5000                       ; roll forward and back 10 times
      for b55 = 1 to 10
            SetHeadingSpeed(0,128)
            Pause 1000
            SetHeadingSpeed(180,128)
            Pause 1000
      next b55
```

red_colour = b0 = 0 = $00 = %00000000

New Enhanced Features for Existing PICAXE Microcontroller Parts

- Second additional program slot for 14M2, 18M2+ and 20M2 parts, effectively doubling their program capacity.
- New variable push/pop commands for X2 parts, enabling any byte variable to be temporarily stored on a 32 level byte stack (push, pop commands).
- New variable pushram/popram commands for X2 parts, enabling the variable block b0-b15 to be stored on a separate 4 level RAM stack (pushram, pushram clear, popram commands).
- New #no_debug directive for all parts, allows instant disabling of all the debug commands within a program
- New '#ifdef simulating' directive for all parts, to conditionally compile additional code only when on-screen simulating.

Quick Glance FAQ

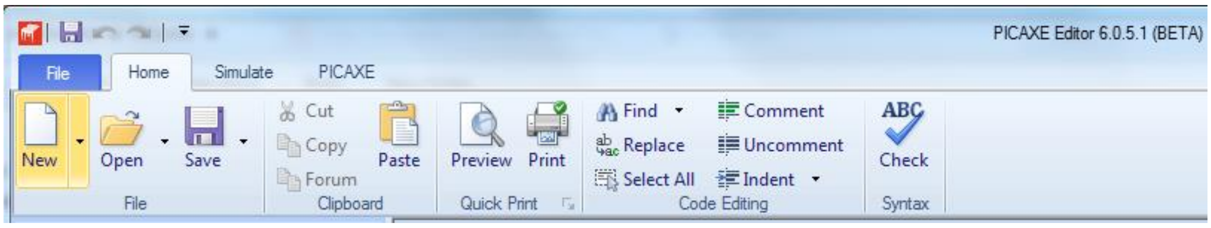
- PE6 will operate on the same Windows versions as PE5, any version of Windows since XP (XP, Vista, 7, 8 - in both 32 and 64 bit versions)
- PE6 supports exactly the same download cables and PICAXE chips as PE5, no hardware changes are required. The same driver is used for the AXE027 USB cable and does not need to be reinstalled if you already use this cable with PE5.
- PE6 and PE5 are completely separate applications, so can safely be installed together on the same computer. BASIC files can be safely shared between both applications.
- PE6 includes a legacy mode that makes it 'look and feel', as far as possible, like PE5 for those who want to stick to the traditional appearance. However we expect most users to enjoy using the new PE6 ribbon interface.
- The PICAXE compilers have always been separate 'plug-in' applications. Therefore the same compiler is used in PE6 as was used in PE5, which means your existing code will compile the same as before. However PE6 also adds the option of a new advanced pre-processor (e.g. to add new features such as to #include one BASIC file within another).
- Logicator flowcharts are now completely merged into PE6, so Logicator as a separate product is no longer required.
- PE6 will always be completely free to download and use.
- PE6 is initially a Windows only application. Once Windows development complete we will look at migrating to Linux/Mac.
- PE5 will be bug maintained for a changeover period until 2015. However no new features or new PICAXE parts will be added to PE5 in the future.

Section 2: Key Features

Menubar and Toolbars

In Legacy mode the traditional menu and toolbar layout is used. Toolbar icon size can be the standard 16x16 pixel or large 32x32 pixel. Menus and toolbars can be docked or floating and positioned as required. Right click over a toolbar to hide/show the various toolbars.

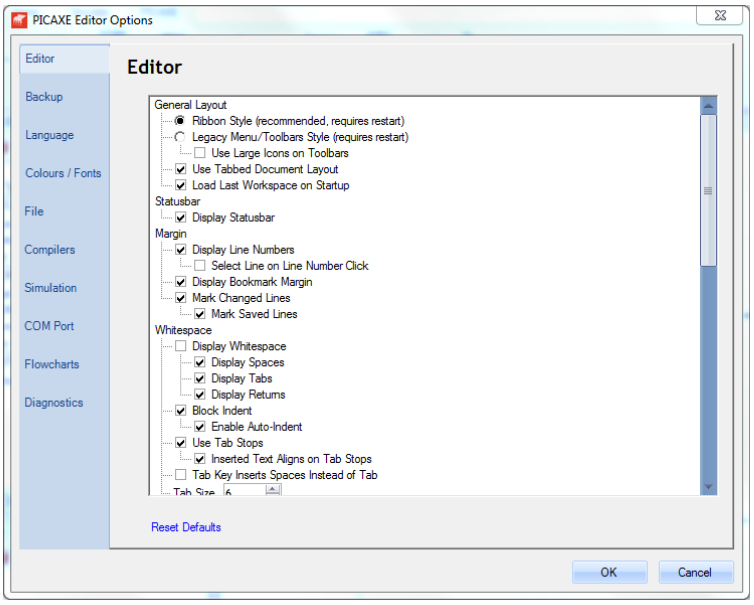
Ribbon



In the new Ribbon mode the three ribbon tabs (Home, Simulation and PICAXE) provide rapid access to all the application features. The QAT (Quick Access Toolbar) is a standard ribbon feature that adds a small toolbar to the titlebar at the top of the application window. The default QAT buttons include undo, redo and save but the QAT is also fully customisable by the end use e.g. you can add a Syntax Check or Program button if you choose.

In Ribbon mode the File menu uses the standard Windows 'backstage' format which provides much greater information than what is available when in Legacy mode – for instance on the Print tab the new Print Preview feature is instantly displayed.

Options



PICAXE Editor Options

Editor

General Layout
- ● Ribbon Style (recommended, requires restart)
- ○ Legacy Menu/Toolbars Style (requires restart)
  - ☐ Use Large Icons on Toolbars
- ☑ Use Tabbed Document Layout
- ☑ Load Last Workspace on Startup

Statusbar
- ☑ Display Statusbar

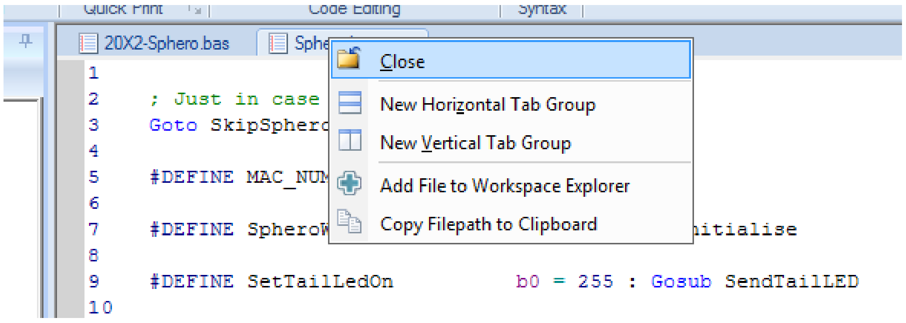
Margin
- ☑ Display Line Numbers
  - ☐ Select Line on Line Number Click
- ☑ Display Bookmark Margin
- ☑ Mark Changed Lines
  - ☑ Mark Saved Lines

Whitespace
- ☐ Display Whitespace
  - ☑ Display Spaces
  - ☑ Display Tabs
  - ☑ Display Returns
- ☑ Block Indent
  - ☑ Enable Auto-Indent
- ☑ Use Tab Stops
  - ☑ Inserted Text Aligns on Tab Stops
- ☐ Tab Key Inserts Spaces Instead of Tab
- Tab Size 6

The Options dialog is accessible from the File>Options menu and includes a much greater range of options that before. It is recommended that all new users take a moment to read through all the available options on each tab, as this gives a very good overview of all the new optional PE6 features.

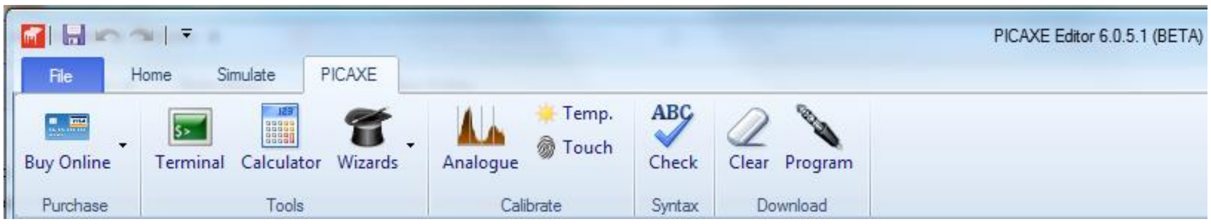
MDI (Multiple Document Interface)

Multiple documents can be either tabbed or floating (cascaded). In tabbed mode multiple vertical and horizontal groups can be arranged as required (simply right click over the tab to trigger a menu that allows you to add new vertical/horizontal tab groups).

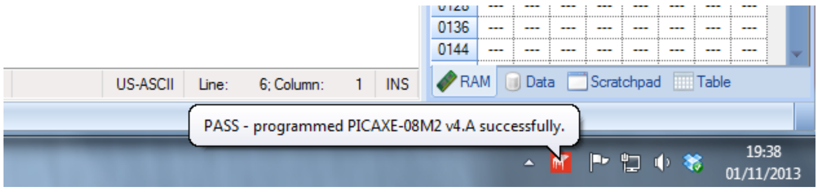


```
; Just in case
Goto SkipSpher

#DEFINE MAC_NUM

#DEFINE SpheroW                                   itialise

#DEFINE SetTailLedOn          b0 = 255 : Gosub SendTailLED
```

Program Downloads



Programming is now performed as a background operation, so it is possible to continue using the software whilst a program (or debug) is in progress. The progress bar is shown in the statusbar (bottom right) and a successful download is also notified via a system tray pop-up.



PASS - programmed PICAXE-08M2 v4.A successfully.

PE6 and Project Workspaces

PE6 now has two modes of operation – 'Simple' and 'Advanced Project Workspaces'.
This mode is selected via the File>Options>Editor tab.

1) Simple Mode.

In simple mode PICAXE programs are generally contained within a single .bas BASIC file or .plf flowchart. However multiple files can still be opened at the same time, and the layout of the visible documents is maintained whenever PE6 is closed / restarted. Individual files are accessed via the File Open/Save/Close menus.

This mode is similar in operation to PICAXE Programming Editor v5.

2) Advanced Project Workspace Mode

A 'project workspace' is a collection of files associated within a project, as well as a container for specific settings for that project (e.g. PICAXE chip type). A workspace should be considered as an 'index' – a method of rapidly accessing all files required within that project.

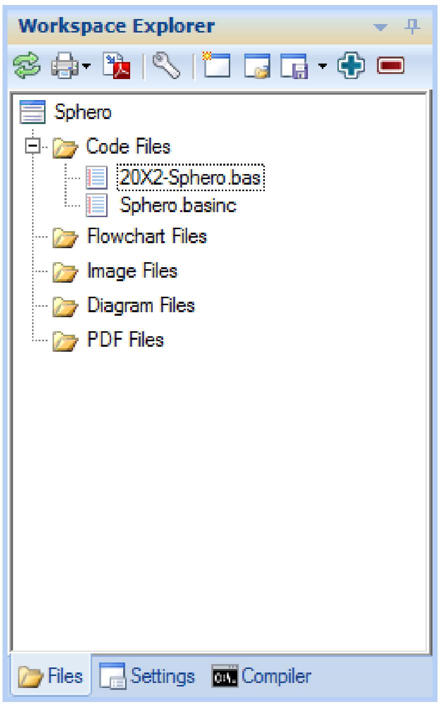
In project workspace mode each project can be allocated its own, separate, layout. This is most useful when each project consist of several files (e.g. a main .bas file, several include .basinc file and even images such as PCB layout or schematic diagram).

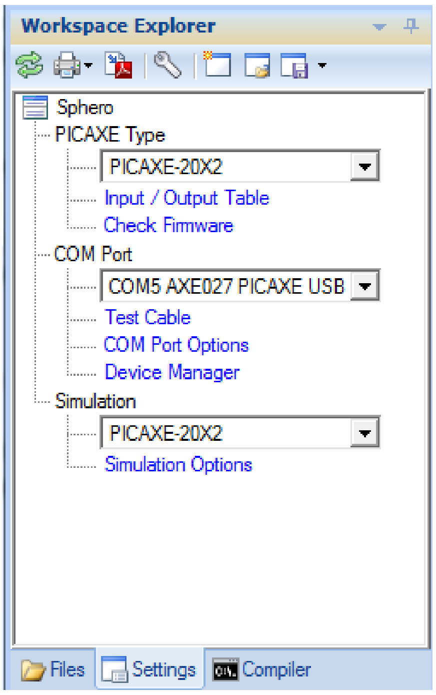
The principle difference to simple mode is that when you open a previous project you open the project workspace (.wsp) file, not individual code files. **Therefore the main action when swapping between various projects is with the *Workspace* Open/Save buttons, not the individual *File* Open/Save buttons.**

The process of opening the Workspace .wsp file then automatically opens all of the other individual files contained in the workspace. Any file associated with the project (displayed within the Files tree on the Workspace Explorer) can also be easily accessed by double clicking on the filename.

Similarly settings such as the PICAXE type, preferred COM port, simulation image etc are all also maintained within the .wsp workspace file. Therefore, for example, when you re-open a Workspace the selected PICAXE type will automatically change to the type previously saved for that particular project.

Workspace Explorer Panel





The Workspace Explorer has three tabs, although the first 'Files' tab is not required in 'Simple' mode and so is only visible when 'Advanced' mode is selected.

The 'Files' tab shows an index of all the files associated with this project workspace. Files can be easily added or removed using the toolbar buttons. Opening a file via the File menu does not automatically add it to the Workspace index, although you can rapidly do this via the 'Add to Workspace' menu that is available on that file's tab.

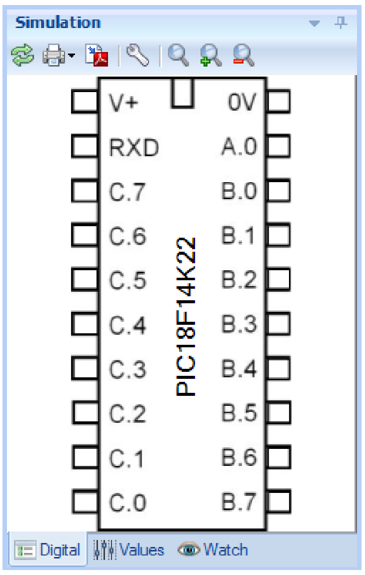
Files are grouped by file extension – BASIC code files, Flowchart files, Image Files (e.g. a PCB layout image or photo), PICAXE Diagram files (e.g. a circuit diagram,) and finally PDF datasheets. Double clicking on a file in the index tree will open it.

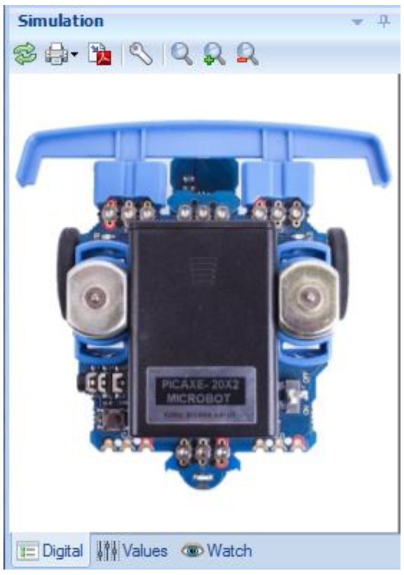
The 'Settings' tab is used to set the PICAXE chip type, COM port and Simulation diagram used by this project. These values are saved within the workspace so that, for instance, the correct PICAXE chip type is automatically selected next time the workspace is opened.

Also included within 'Settings' are various common feature short cut links. One new feature available here is the 'Input/Output Table', which is a commonly requested feature that instantly displays the selected PICAXE chip's pinout and pin feature summary directly within the Editor.

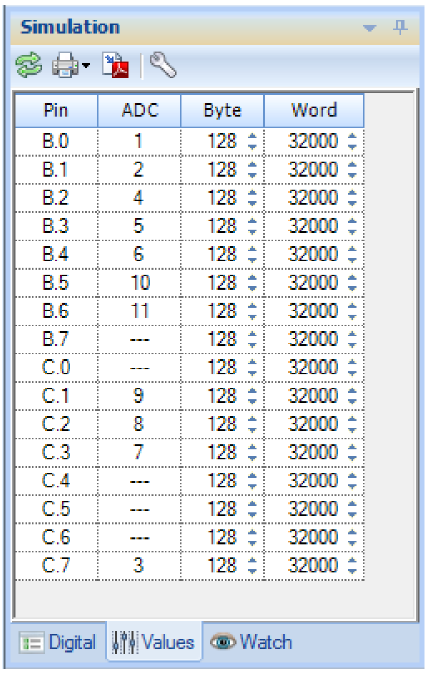
The 'Compiler' tab displays the raw feedback messages from the PICAXE compilers and pre-processor.

Simulation Panel



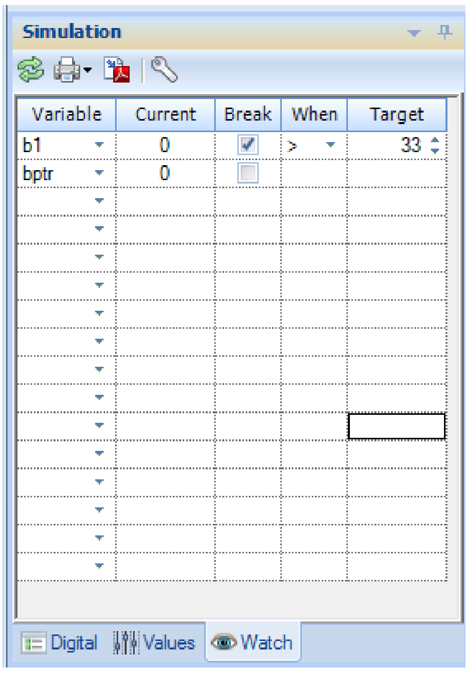


The simulation panel displays the interactive Simulation diagram. This defaults to the pinout of the particular PICAXE chip selected, but can be changed to an interactive project board or even a photo of your own project (users can build their own interactive simulations if required). The simulation diagram shows the state of the outputs during a simulation, and also allows inputs to be changed via mouse clicks over the appropriate input. If desired the simulation diagram can be undocked into a floating window if desired via a right click.



Pin	ADC	Byte	Word
B.0	1	128	32000
B.1	2	128	32000
B.2	4	128	32000
B.3	5	128	32000
B.4	6	128	32000
B.5	10	128	32000
B.6	11	128	32000
B.7	---	128	32000
C.0	---	128	32000
C.1	9	128	32000
C.2	8	128	32000
C.3	7	128	32000
C.4	---	128	32000
C.5	---	128	32000
C.6	---	128	32000
C.7	3	128	32000

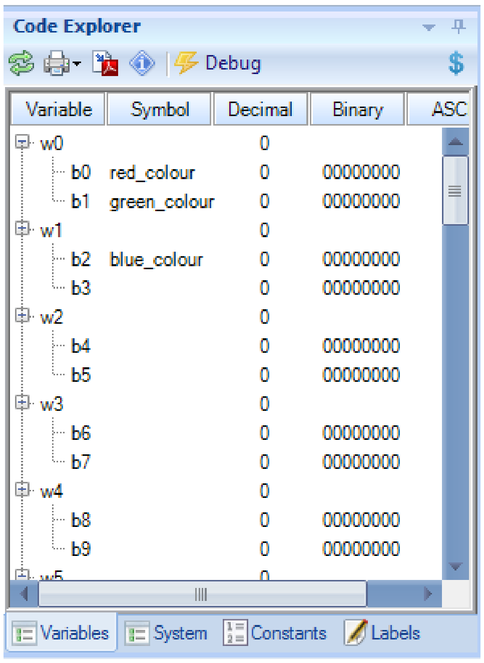
The second ‘values’ tab allows input during a simulation to commands such as ‘readadc’ and ‘pulsin’ etc. Byte orientated commands (e.g. readadc) use the value in the byte column, word orientated commands (e.g. pulsin) use the value in the word column. This new byte/word system overcomes the limitations of the single ’generic’ input box within PE5.



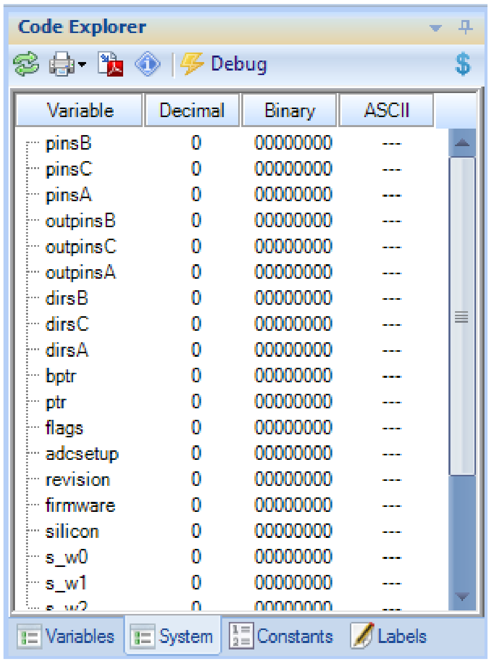
Variable	Current	Break	When	Target
b1	0	✓	>	33
bptr	0	☐		

The third ‘Watch’ tab is a new feature that allows breakpoints to be set on any of the RAM or System RAM variables. It is also useful for comparing particular values side by side as a simulation runs e.g. ‘b23’ and ‘pinsB’ can be located next to each other for comparison.

Code Explorer Panel



Variable	Symbol	Decimal	Binary	ASC
w0		0		
b0	red_colour	0	00000000	
b1	green_colour	0	00000000	
w1		0		
b2	blue_colour	0	00000000	
b3		0	00000000	
w2		0		
b4		0	00000000	
b5		0	00000000	
w3		0		
b6		0	00000000	
b7		0	00000000	
w4		0		
b8		0	00000000	
b9		0	00000000	
w5		0		



Variable	Decimal	Binary	ASCII
pinsB	0	00000000	---
pinsC	0	00000000	---
pinsA	0	00000000	---
outpinsB	0	00000000	---
outpinsC	0	00000000	---
outpinsA	0	00000000	---
dirsB	0	00000000	---
dirsC	0	00000000	---
dirsA	0	00000000	---
bptr	0	00000000	---
ptr	0	00000000	---
flags	0	00000000	---
adcsetup	0	00000000	---
revision	0	00000000	---
firmware	0	00000000	---
silicon	0	00000000	---
s_w0	0	00000000	---
s_w1	0	00000000	---
s_w2	0	00000000	---

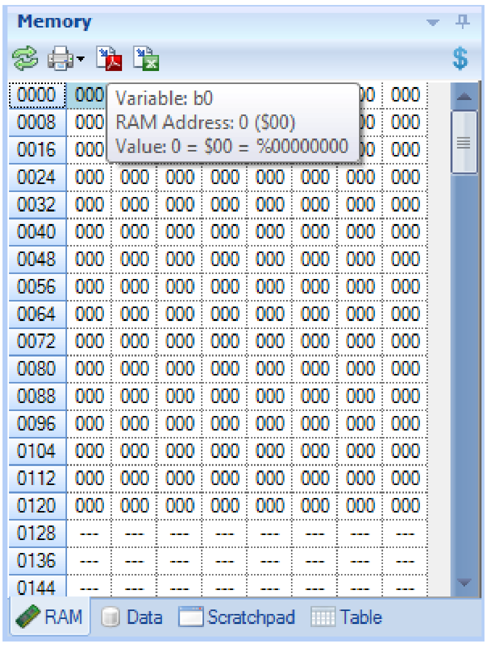
The code explorer panel shows the values of the variables, main system variables and user defined constants. The labels used within a program are also displayed.

In PE5 the ‘debug’ window was a modal window that means that no program editing can be carried out whilst a debug session is in process. In PE6 the debug feature has been fully integrated into the Code Explorer so that it now acts as a background task. Simply click the debug button to start the debug session (or download a program containing the debug command) and then carry on working as normal.

In addition the new #no_debug directive can be used to disable all debug commands within your program, so that they are simply ignored and not included within any download.

If a variable clash is identified within a program (e.g. use of both b1 and w0 as two different symbols) a warning summary can be printed from the ‘Variable Clash’ button on the toolbar.

Memory Panel



Variable: b0
RAM Address: 0 ($00)
Value: 0 = $00 = %00000000

0000	000						0	000
0008	000						0	000
0016	000						0	000
0024	000	000	000	000	000	000	000	000
0032	000	000	000	000	000	000	000	000
0040	000	000	000	000	000	000	000	000
0048	000	000	000	000	000	000	000	000
0056	000	000	000	000	000	000	000	000
0064	000	000	000	000	000	000	000	000
0072	000	000	000	000	000	000	000	000
0080	000	000	000	000	000	000	000	000
0088	000	000	000	000	000	000	000	000
0096	000	000	000	000	000	000	000	000
0104	000	000	000	000	000	000	000	000
0112	000	000	000	000	000	000	000	000
0120	000	000	000	000	000	000	000	000
0128	---	---	---	---	---	---	---	---
0136	---	---	---	---	---	---	---	---
0144	---	---	---	---	---	---	---	---

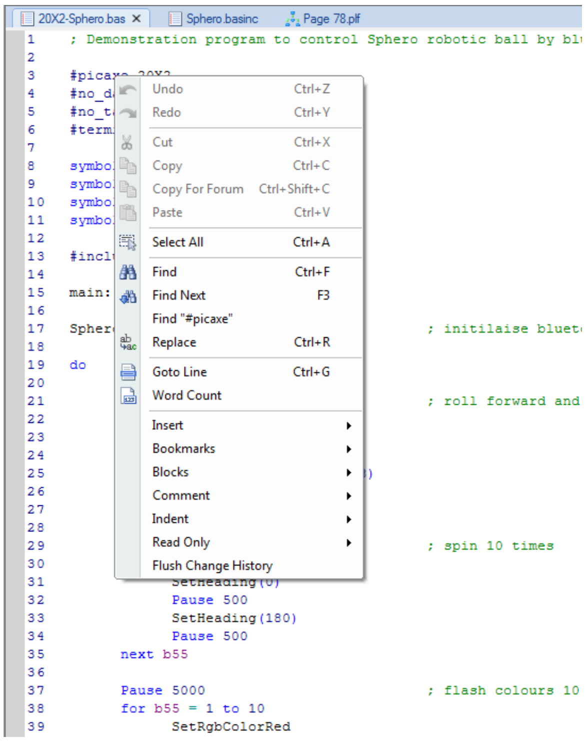
The memory panel shows all values of the RAM, EEPROM data, scratchpad RAM and Table memory areas. The display defaults to decimal, but may be converted to hex by clicking the '$' button.

On the RAM and scratchpad grids one of the cells will always be highlighted in a darker colour. This is the cell currently pointed to by the bptr/ptr variable, and so the value displayed in that highlighted cell is the current value that will be accessed by @bptr/@ptr.

During a simulation you can double click on any of the RAM values to change its value.

Code Editor

The code editor includes many new and improved features. Almost all features may be enabled/disabled as desired via the File>Options dialog.



```
; Demonstration program to control Sphero robotic ball by bl

#picax
#no_d
#no_t
#term

symbo
symbo
symbo
symbo

#incl

main:

Spher                                          ; initilaise bluet

do

                                               ; roll forward and



                                          )



                                               ; spin 10 times

                SetHeading(0)
                Pause 500
                SetHeading(180)
                Pause 500
        next b55

        Pause 5000                             ; flash colours 10
        for b55 = 1 to 10
                SetRgbColorRed
```

Right click whilst editing to display the extended context menu, which provides rapid access to the main editing features, including new editor features such as block comment/uncomment.

Margin
The margin displays the line number, bookmarks and breakpoints. An optional coloured vertical band also highlights changes made to the code since last opened and since last saved.

Interactive Tooltips
Interactive tips now provide additional new information about all types of BASIC program content.

- Commands – the tooltip provides basic command structure
- Variables – the tooltip shows the current variable value in decimal, hex and binary
- Constants – the tooltip shows the constant value in decimal, hex and binary.
- Numbers - the tooltip shows the value in other formats (e.g. decimal and binary for a hex number)
- Strings – the tooltip shows the ASCII values of the characters in that string
- Pins – the tooltip shows the main pin functions

Improved scrollbars and intellimouse support assist navigation.

Improved Find and Replace features, including bookmark of all found instances

Breakpoints and bookmarks are now saved alongside the file.

Many additional keyboard shortcuts are now supported e.g. both Ctrl+V and Shift+Ins for ‘Paste’

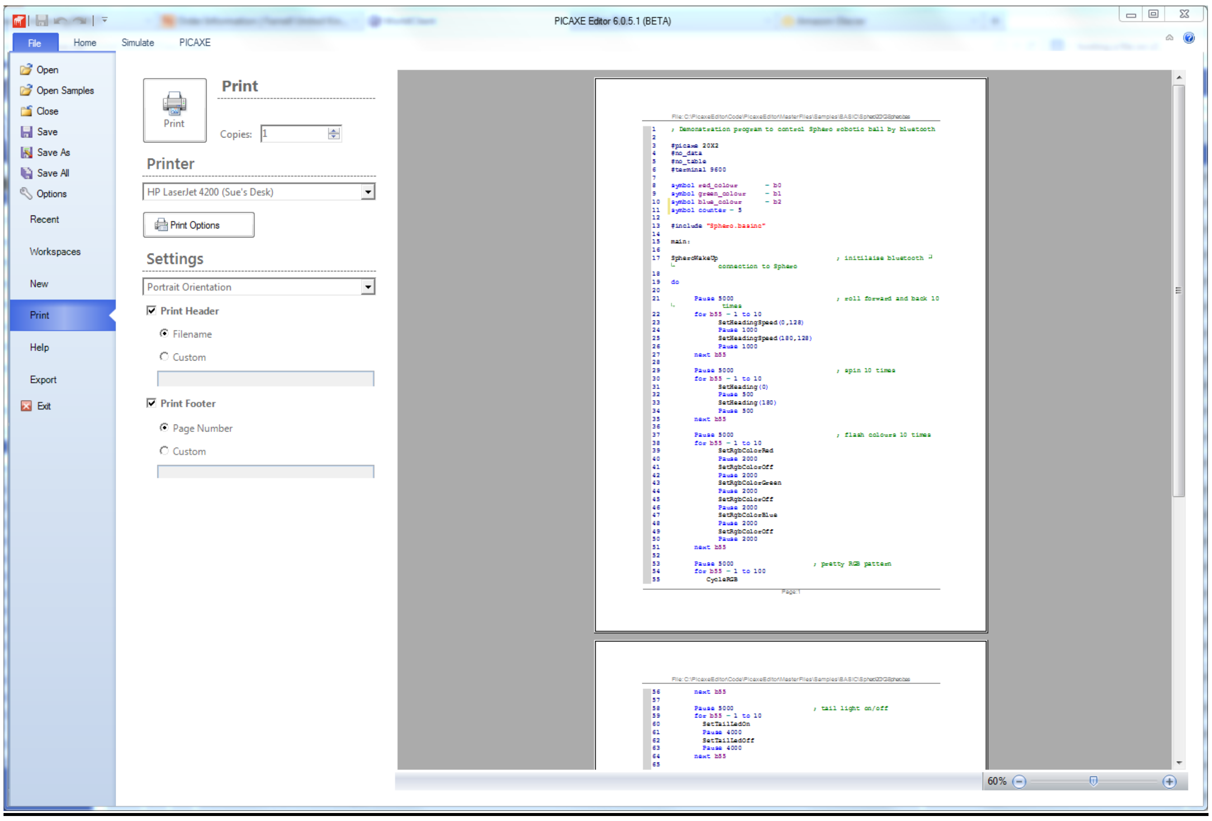
Image Viewer

The image window can be used to display most common image types (jpg, png etc) which will typically be photos of the project, PCB layouts, schematics etc.

Diagram Viewer

The diagram window is used to display PICAXE interactive diagrams. This may be a simulation diagram, breadboard layout or circuit diagram etc. See the PICAXE Diagram Builder (under development) for more details.

Printing and PDF Export



Print

Copies: 1

Printer

HP LaserJet 4200 (Sue's Desk)

Settings

Portrait Orientation

[✓] Print Header

(●) Filename

( ) Custom

[✓] Print Footer

(●) Page Number

( ) Custom

File: C:\PicaxeEditor\Code\PicaxeEditor\MasterFiles\Samples\BASIC\Sphero20X2Sphero.bas

```
; Demonstration program to control Sphero robotic ball by bluetooth

#picaxe 20X2
#no_data
#no_table
#terminal 9600

symbol red_colour       = b0
symbol green_colour     = b1
symbol blue_colour      = b2
symbol counter = 5

#include "Sphero.basinc"

main:

SpheroWakeUp                              ; initilaise bluetooth 
           connection to Sphero

do

      Pause 5000                          ; roll forward and back 10
           times
      for b55 = 1 to 10
            SetHeadingSpeed(0,128)
            Pause 1000
            SetHeadingSpeed(180,128)
            Pause 1000
      next b55

      Pause 5000                          ; spin 10 times
      for b55 = 1 to 10
            SetHeading(0)
            Pause 500
            SetHeading(180)
            Pause 500
      next b55

      Pause 5000                          ; flash colours 10 times
      for b55 = 1 to 10
            SetRgbColorRed
            Pause 2000
            SetRgbColorOff
            Pause 2000
            SetRgbColorGreen
            Pause 2000
            SetRgbColorOff
            Pause 2000
            SetRgbColorBlue
            Pause 2000
            SetRgbColorOff
            Pause 2000
      next b55

      Pause 5000                    ; pretty RGB pattern
      for b55 = 1 to 100
         CycleRGB
```

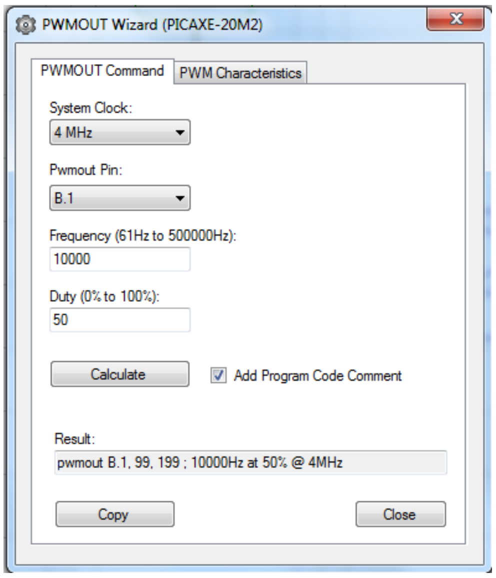
Page:1

File: C:\PicaxeEditor\Code\PicaxeEditor\MasterFiles\Samples\BASIC\Sphero20X2Sphero.bas

```
      next b55

      Pause 5000                    ; tail light on/off
      for b55 = 1 to 10
        SetTailLedOn
        Pause 4000
        SetTailLedOff
        Pause 4000
      next b55

```

The printing, print preview and embedded PDF export features have been greatly improved in PE6. You can now export/print almost any area of the screen, including the workspaces and panels.

Wizards



System Clock: 4 MHz

Pwmout Pin: B.1

Frequency (61Hz to 500000Hz): 10000

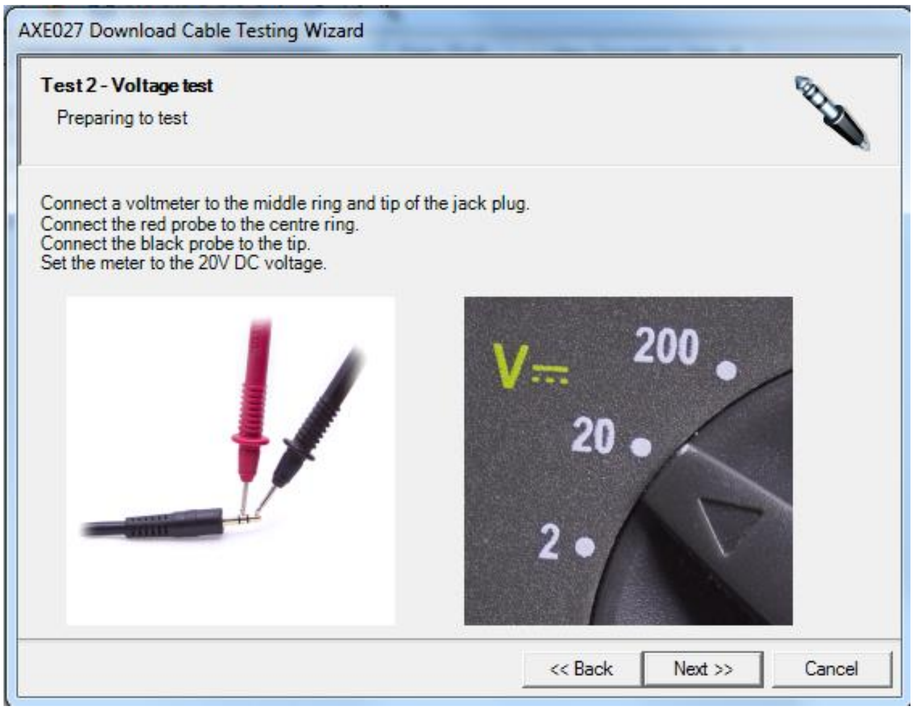
Duty (0% to 100%): 50

☑ Add Program Code Comment

Result:

```
pwmout B.1, 99, 199 ; 10000Hz at 50% @ 4MHz
```

The code wizards now use an external plug-in module structure, which allows new wizards to be much more easily added as and when required. All wizards have been re-created from scratch, and several such as the 'pwmout' wizard now have extra additional and/or enhanced features.



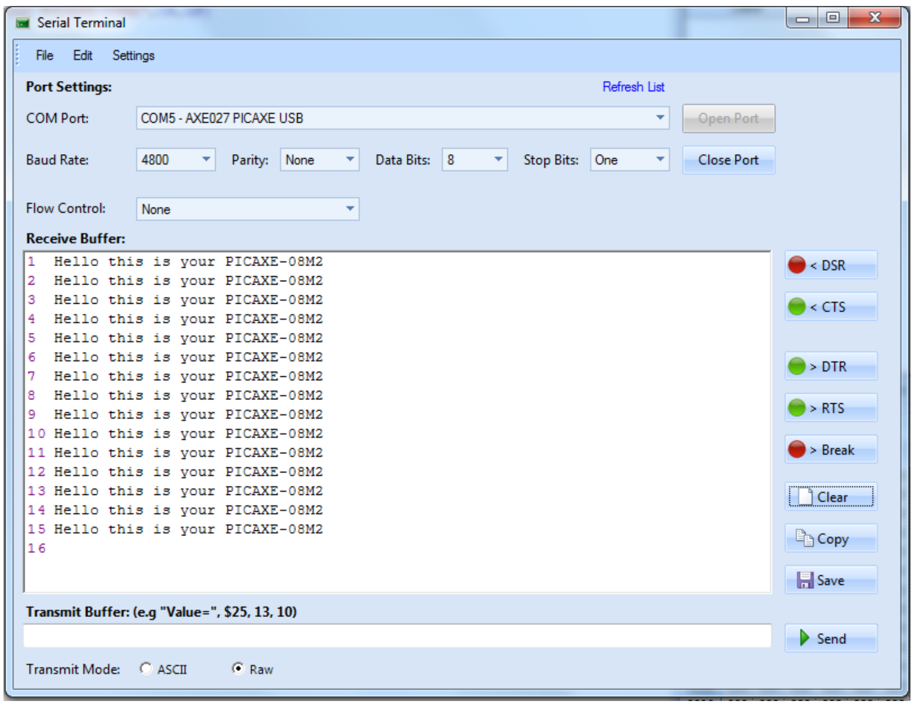
Test 2 - Voltage test

Preparing to test

Connect a voltmeter to the middle ring and tip of the jack plug.
Connect the red probe to the centre ring.
Connect the black probe to the tip.
Set the meter to the 20V DC voltage.

New wizards such as the AXE027 trouble shooter are now also included.

Serial Terminal



Port Settings:
COM Port: COM5 - AXE027 PICAXE USB
Baud Rate: 4800 Parity: None Data Bits: 8 Stop Bits: One
Flow Control: None

Receive Buffer:

```
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
Hello this is your PICAXE-08M2
```

Transmit Buffer: (e.g "Value=", $25, 13, 10)

Transmit Mode: ( ) ASCII [✓ Raw]

The Serial Terminal has been significantly enhanced and improved.
Key new features include:

- Can use separate COM port from PICAXE programming port on multiple port machines
- All port settings such as handshaking and stop bits now available
- Control signals such as CTS, RTS etc. now visible
- Non-modal, so can be used at same time as main Editor
- Hex, decimal or ASCII display modes with colour coding
- Display of non-printing ASCII characters received
- Control characters can be optionally used to navigate about the receive box

The output (send) text box has also been remodelled. In PE5 the output sent was always ASCII, which confused some people when sending numbers e.g. is 23 the number 23 or the two ASCII characters "2"+"3"?

In PE6 a new 'raw mode' allows the output box text to be formatted exactly the same as within PICAXE serout and serin commands, so raw data and strings can now be much more simply mixed and sent together e.g.

"hello",13,10

Note that to send ASCII strings in PE6 (i.e. to match the old PE5 behaviour) select the 'ASCII' mode instead (or simply use raw mode with the text wrapped within double quotes).

Received serial input can be saved, printed or exported to PDF.

Note that on-screen simulations now also use exactly the same Terminal window, so behaviour of 'simulated' serial much more closely matches 'real-life' serial.

Analogue Sensor Calibration

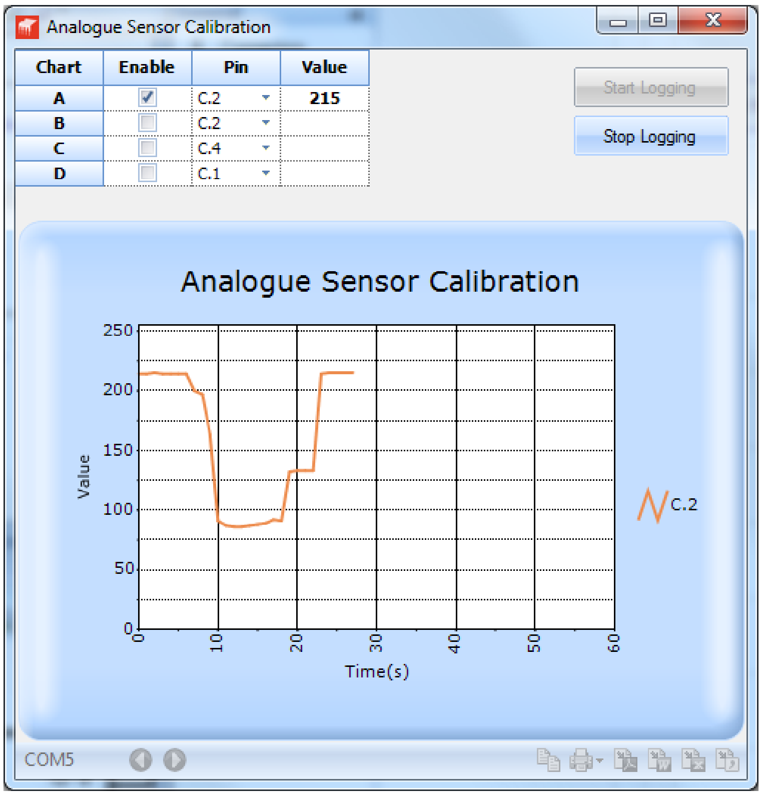
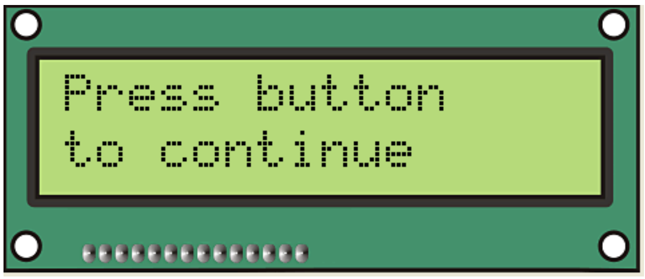


Chart	Enable	Pin	Value
A	✓	C.2	215
B		C.2	
C		C.4	
D		C.1	

The sensor calibration feature has now also been extensively updated and can show a real time graph of the analogue values whilst experimenting with the calibration. Result scan be exported as graphs, PDF or Excel spreadsheets.

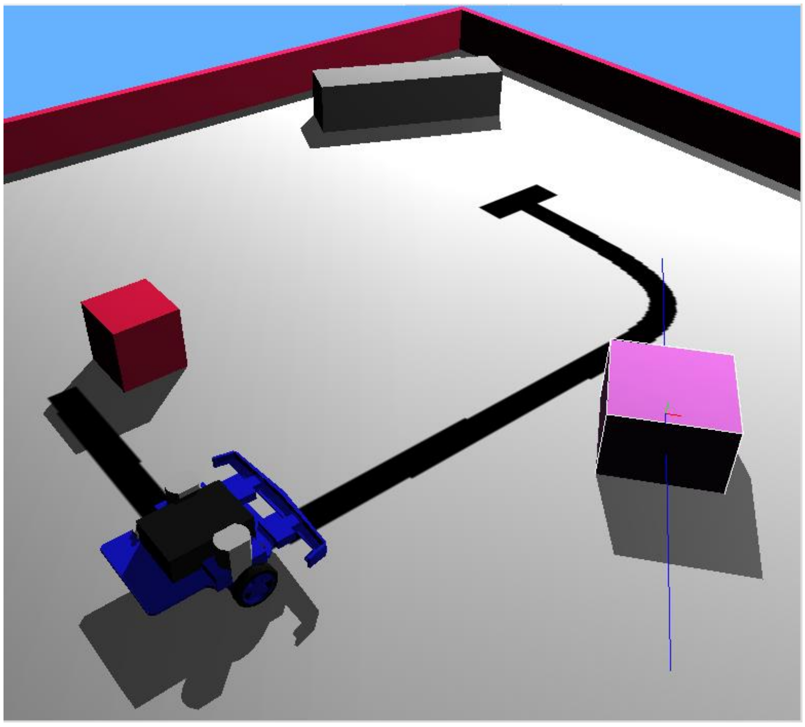
Section 3: Simulation Engine

The on-screen simulation engine has been re-engineered to make it more efficient and to more closely mimic real-life devices.



Key new features include:

- Configurable simulation 'diagram' for different input/output layouts
- Simulation of multiple file programs (when used with #INCLUDE)
- Better support for multiple analogue inputs and multiple general inputs (pulsin, count etc.)
- Improved variable watch panel and conditional breakpoints
- Serial simulation uses identical Serial Terminal to a real device
- Improved sound and tune simulation
- LCD/OLED and infra-red controller simulation more realistic
- All panels can be docked or floating and positioned as required.
- Non-graphical 'fast run' to more closely mimic real device
- Support for connected PICAXE chip, so that real input/outputs are activated via an on-screen simulation
- Real-time links to third party circuit simulation and 3D modelling software e.g. Webots for a 3D simulation of a moving PICAXE robot
- Flowcharts use exactly the same simulation engine as BASIC files.


Webots 3D model of BOT120

<u>Section 4: The PICAXE Pre-Processor</u>

The main purposes of the new PICAXE Pre Processor are
1) To allow BASIC programs to be split over multiple files
2) To allow macros (user defined functions)
3) To allow multiple word string replacement e.g. myhighC(x) = high portC x

The pre-processor takes one or more input files, combines them together, substitutes and processes any macros, and then generates a single output file which is then passed to the standard PICAXE compiler. If desired end users can view the final pre-processor output code via the File>Options>Diagnostics tab.

Preprocessor Directives

Directives have a hash/pound (#) symbol as the first non-blank character of a line, and are immediately followed by the directive keyword.

There are two main types of directive, those used by the pre-processor and those used by the compiler. The pre-processor simply ignores compiler directives.

The following is a list of all the pre-processor directives.

#PICAXE	Define the PICAXE type. This must match the Workspace Explorer setting.
#INCLUDE	Include and merge a specified file, which generally has a .basinc file extension.
#DEFINE	Define a single line symbol
#MACRO/#ENDMACRO	Define a multiline symbol or macro
#UNDEF	Undefine a previously defined symbol/macro
#IF	Process next block if symbol/calculation is true
#IFDEF	Process next block if symbol is defined
#IFNDEF	Process next block if symbol is not defined
#ELSEIF	Process next block if symbol/calculation is true
#ELSEIFDEF	Process next block if symbol is defined
#ELSE	Process next block if previous blocks have not been processed.
#ENDIF	Terminate a conditional block
#REM	Start commenting a section of code
#ENDREM	Finish commenting of a section of code
#ERROR	Force an error at the current position

Compiler Directives

The following is a list of compiler directives. These directives are simply ignored by the pre-processor.

Compiler Directives:

#NO_DEBUG	Do not output any debug commands within the program.
#NO_END	Do not add the automatic END command at end of file
#TITLE	Set a title for the remote download tool
#REVISION	Set a file version number

Compiler/Downloader module directives:

#SLOT	Define the target program slot
#COM	Select the COM port for downloading (overrides the Workspace Explorer)
#NO_DATA	Do not download EEPROM data
#NO_TABLE	Do not download TABLE data
#TERMINAL	Open the terminal at a specified baud rate
#FREQ	Set freq for obsolete old parts

Presentation Directive:

#REGION/#ENDREGION	Define a collapsing block region

Deprecated in PE6

#GOSUBS	- Now select special compiler via File>Options>Compilers
#SIMTASK	- Now select Simulation on Workspace Explorer
#SIMSPEED	- Now simply use the simulation speed slider bottom right of statusbar

Pre-processor Variable String Substitutions

The following predefined pre-processor string variables will substitute as follows during pre-processing. Time and dates are based upon the computer's system clock.

ppp_date	"YYYY-MM-DD"
ppp_date_uk	"DD-MM-YYYY"
ppp_date_us	"MM-DD-YYYY"
ppp_datetime	"YYYY-MM-DD HH:MM:SS"
ppp_time	"HH:MM:SS"
ppp_filename	The short filename of the main BASIC inputfile
ppp_filepath	The full filename and path of the main BASIC inputfile

Example of use:

```
    sertxd (“Filename: ”, ppp_filename)
    sertxd (“Downloaded: ”, ppp_datetime)
```

#INCLUDE “filename.basinc”

The #INCLUDE directive opens and processes a file, then continues processing the original file. A file opened through the #INCLUDE directive can itself contain other #INCLUDE directives. There is no limit to the number of files that can be nested.

The filename must be enclosed within double quotes and can use an absolute or relative filepath. If the path is relative the pre-processor will attempt to locate the file in these folders in this order:

1) the folder of the main top-level file
2) each folder identified by the Windows %Path% Environment Variable

Included files are normally named with a .basinc extension, but may also be .bas files. There is no difference between the format of the files apart from the filename extension.

The advantages of using the .basinc extension for include files are:

1) PE6 disables the Syntax Check/Program, buttons on this file type, so that you don’t accidentally try to compile an include file by itself
2) PE6 automatically saves any open .basinc files on each compile of the main program. This is to ensure that any changes made on screen to the included file are correctly included within the compile.
3) Other PICAXE users will recognise the file as an include file

Included files must be saved as ASCII or UTF-8 Unicode files with a line ending type CR or CR-LF (not LF only).

As the compiler processes ’top-to-bottom’ on the pre-processor output file it is necessary to take care to ‘skip’ the #INCLUDE file (if necessary) when it is included at the top of a program.

```
Reset_here:  Goto Init          ; jump over include file

#INCLUDE     “sphero.basinc”

Init:                           ; start program here
```

#DEFINE and #MACRO

#DEFINE is used to define directive symbols that are then used within the pre-processing.
Do not confuse with 'symbol xxx = yyy' type symbols which are used by the compiler within the main program. The pre-processor cannot use or even recognise 'symbol xxx = ' defined symbols.

#MACRO/#ENDMACRO (or #ENDM if you are lazy typist) is simply the multiline version of #DEFINE

Examples of use:

1) Simple symbol definition for use with #IFDEF/#IFNDEF:

```
#DEFINE SPHERO
```

2) String substitution:

```
#DEFINE MAGIC_NUMBER            "0006664A3BFB"
#DEFINE SpheroWakeUp            Gosub Sphero_Initialise
#DEFINE SetBackLedOn            b0 = 255 : Gosub SendBackLED
#DEFINE SetBackLedOff           b0 = 0 : Gosub SendBackLED
```

3) Numeric value substitution:

```
#DEFINE PROGRAM_VERSION         22
#DEFINE PORT_VALUE              %10101010
#DEFINE FALSE                   0
#DEFINE TRUE                    1
```

4) Substitution of another macro

```
#DEFINE SetRgbColorOff          SetRgbColor(0, 0, 0)
#DEFINE SetRgbColorRed          SetRgbColor(255, 0, 0)
#DEFINE SetRgbColorGreen        SetRgbColor(0, 255, 0)
#DEFINE SetRgbColorBlue         SetRgbColor(0, 0, 255)
```

5) Passing parameters that are then used with the substitution

```
#MACRO SetRgbColor(R, G, B)
      b0 = R
      b1 = G
      b2 = B
      Gosub SendRGB
#ENDMACRO

#DEFINE SetHeading(H)           w0 = H : Gosub SendHeading

#MACRO SetHeadingSpeed(H, S)
      w1 = H
      b0 = S
      Gosub SendHeadingSpeed
#ENDMACRO
```

Parameters can be defined using any word that starts with a letter and is not already a BASIC keyword.

Conditional processing.

Processing can be conditional upon use of the #IFDEF/#IFNDEF/#IF directives.

#IFDEF symbol	Process if the symbol has been defined
#IFNDEF symbol	Process if the symbol has not been defined
#IF symbol	Process if the symbol has a true (not 0) numeric value
#IF symbol > constant	Process if the numeric constant calculation is true The supported operators are =, <>, >, >=, <, <=

Multiple tests can be made using the #ELSEIF and #ELSEIFDEF directives, and a single optional #ELSE may also be used before the #ENDIF e.g.

```
#IF symbol > 10

#ELSEIF symbol > 5

#ELSE

#ENDIF
```

or using defined symbols

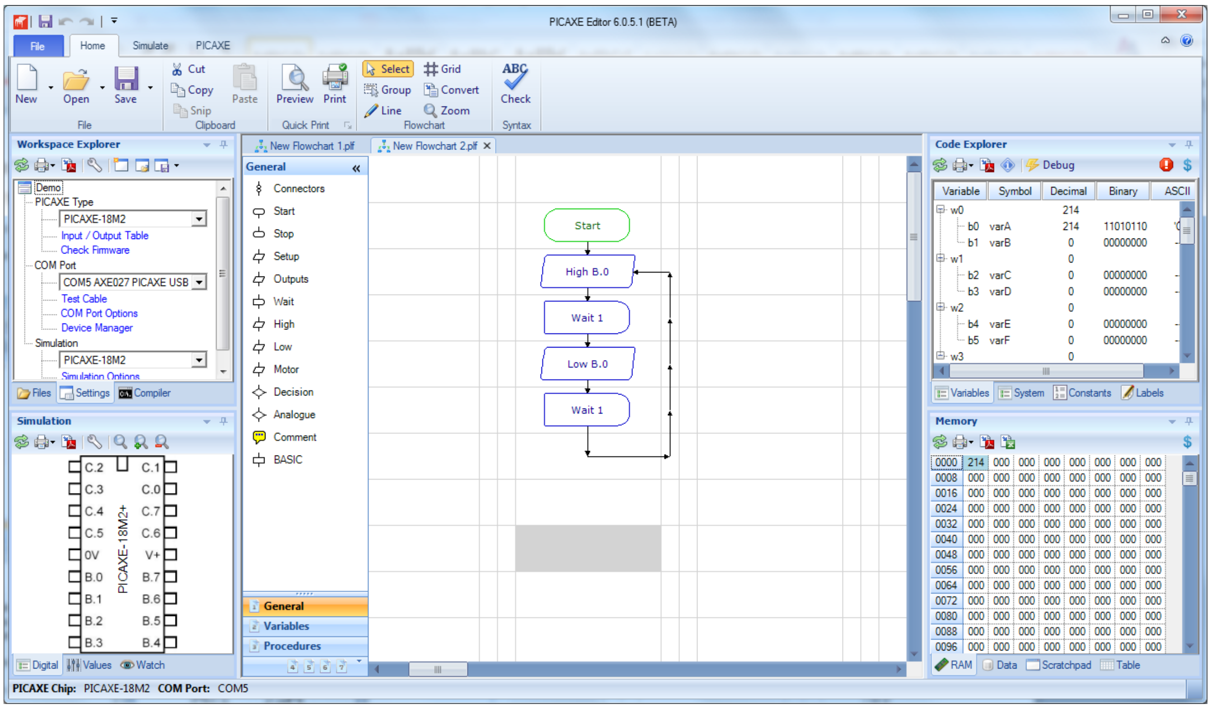
```
#IFDEF symbol1

#ELSEIFDEF symbol2

#ELSEIFNDEF symbol3

#ELSE

#ENDIF
```

Forced Errors

#ERROR “This code is incomplete!”

The #ERROR forces the pre-processor to stop and output the error message.

Section 5: Flowcharts in PE6



Demo
PICAXE Type
PICAXE-18M2
Input / Output Table
Check Firmware
COM Port
COM5 AXE027 PICAXE USB
Test Cable
COM Port Options
Device Manager
Simulation
PICAXE-18M2
Simulation Options

Variable	Symbol	Decimal	Binary	ASCII
w0		214		
b0	varA	214	11010110	[illegible]
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		

0000	214	000	000	000	000	000	000	000
0008	000	000	000	000	000	000	000	000
0016	000	000	000	000	000	000	000	000
0024	000	000	000	000	000	000	000	000
0032	000	000	000	000	000	000	000	000
0040	000	000	000	000	000	000	000	000
0048	000	000	000	000	000	000	000	000
0056	000	000	000	000	000	000	000	000
0064	000	000	000	000	000	000	000	000
0072	000	000	000	000	000	000	000	000
0080	000	000	000	000	000	000	000	000
0088	000	000	000	000	000	000	000	000
0096	000	000	000	000	000	000	000	000

PICAXE Chip: PICAXE-18M2 **COM Port:** COM5

PE5 contained an obsolete flowcharting module that has been superseded for several years by the educationally popular 'Logicator for PIC micros' flowcharting software. However, as Logicator was previously a completely separate application, it also required time-consuming separate development and maintenance.

Therefore Logicator has now been completely merged into PE6, so the single application handles both methods of PICAXE code development - BASIC programming and flowcharts. There is no charge for PE6 so both flowchart programming and BASIC programming modes are completely free to use.

This improvement also allows the flowcharts to be simulated using exactly the same simulation engine as BASIC programs. This provides a more accurate simulation of flowcharts and also allows simulation of any flowchart cells, including generic BASIC cells and user defined cells.

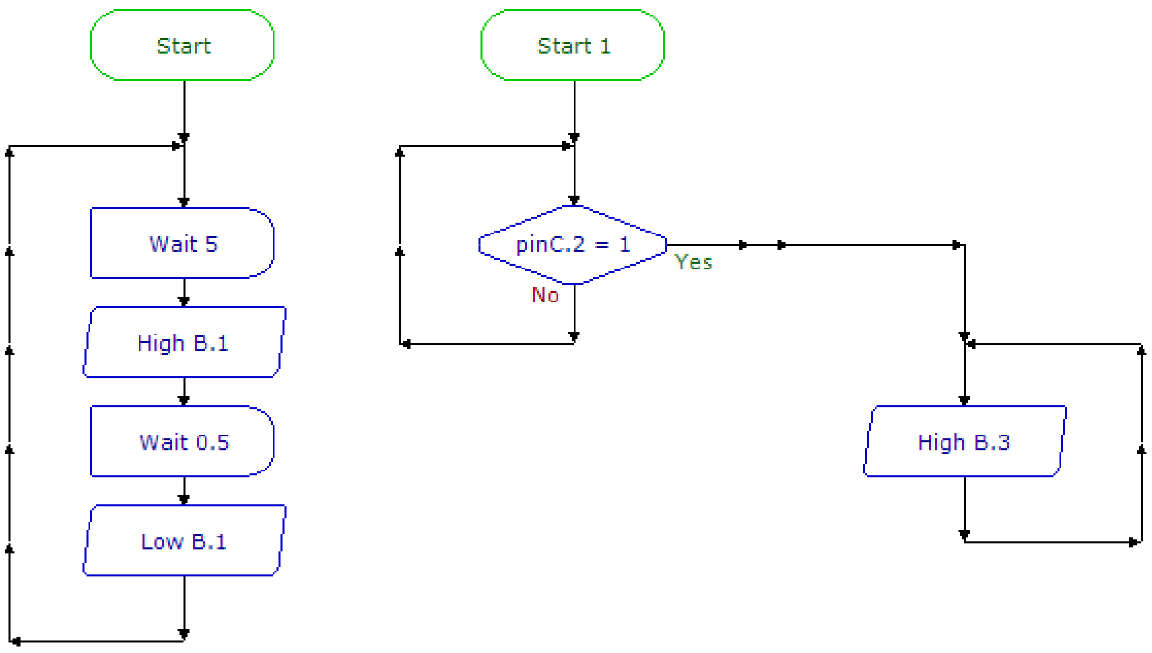
Flowcharts can now also make use of all PICAXE features, in particular configurable i/o pins on M2 and X2 parts, additional variables and many more ADC analogue input channels are now available. The user can now use the new flowchart 'setup' cell to set which pins are inputs and which are outputs on all applicable PICAXE parts.

The flowchart cell toolbox is now also fully configurable, so users can show/hide different command cells depending on the age/experience of the students using the application. If desired users can also create their own, completely new, flowchart cells (e.g. for a particular type of sensor) and then add them to the toolbox. This new cell will also be fully simulated.

Finally the user interface has been completely refreshed with easier line drawing, line deleting and group selection features. Users can also select the colour scheme used with the flowchart.

New flowchart feature summary

- Configure input/output pins on all supported PICAXE types
- More accurate simulation engine supports all command cells, including generic BASIC cells.
- Improved line drawing and deletion
- Improved group selection for cut and paste.
- Multiple flowcharts can be opened simultaneously
- Improved serial terminal, debug and analogue sensor calibration wizards.
- Now supports up to 8 parallel tasks
- New breakpoints and variable watch features
- User selectable colour schemes
- User configurable cell toolbox (hide/display different cell commands)
- User can add completely new 'user defined' command cells
- All command cells now fully localisable for all languages
- Simulation links to 3rd party circuit simulators and 3D models.


Start
Wait 5
High B.1
Wait 0.5
Low B.1
Start 1
pinC.2 = 1
Yes
No
High B.3