

How to Debug

Introduction

Debugging in Small Basic is the process of analysing a program to detect and fix errors or improve functionality in some way.

In order to debug a program it must first compile and run with no syntax errors in the source code, so fix these first.

Using standard Small Basic the techniques to debug a program mainly rely on adding *TextWindow.WriteLine* commands or some other way to see the value of variables at key points in the program. Other commands that can be useful are *GraphicsWindow.Title* (write simple data to the GraphicsWindow title bar), *Program.Delay* (to slow a dynamic program to see what is happening on the screen) or *Sound.PlayClickAndWait* (to give an audible indication that some piece of code was executed).

This document is about extended debugging using SB-Prime.

Overview

Debugging runs the program as normal, except that the program can be paused at specific lines of your code.

When the program is paused, the current line is highlighted with a yellow background colour. Also, when paused, the values of variables can be checked or even changed.

When paused, the program can also be advanced line-by-line so you can see the path taken, for example which branch of an If statement is taken.

As well as advancing line by line you can step over or out of subroutines. This means it can be easier to get to a point in your code where something doesn't seem to work as expected.

Another way to get the program to pause at a point you want is by setting break-points. These can be set at any line in the code and the program will pause when a break-point line is reached.

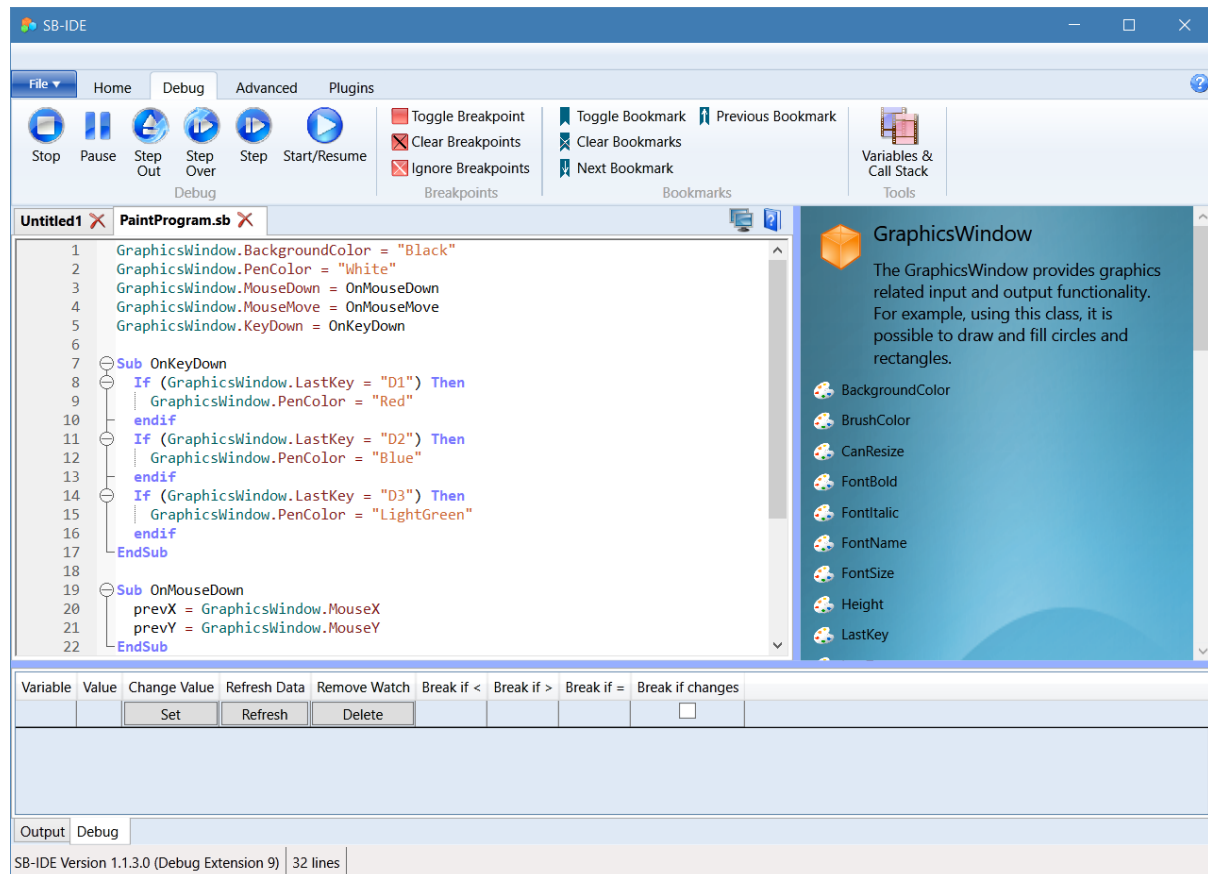
Finally, it is possible to pause the program when the value of a variable changes, or becomes greater than, less than or equal to a certain value. So for example, you want the program to pause when a game score is at a value that is causing problems.

So, in summary debugging allows you to trace the route through a program, checking variables as it proceeds, ensuring it is behaving as you want.

Debug Ribbon Bar

To debug your program, first make sure it compiles and runs. This example shows the PaintProgram.sb sample that comes with Small Basic.

Next select the Debug ribbon bar.



When the debug ribbon bar is selected, the Debug tab at the bottom of the IDE is also selected. Errors and warnings will continue to be sent to the Output tab, so this can also be checked if any errors occur.

For example, you can only debug one program at a time. Occasionally if a program crashes you may have to manually kill the program process, however usually pressing Stop will reset everything to start a new debugging run.

Various commands have function key shortcuts, such as F6 to start debugging or F9 to toggle break-points. The function keys are shown in the tooltip for buttons where they are available.

Run and Step Commands



Start or resume running the program. You can interact normally with the program only while it is running. The program will continue running until it is paused.



Stop and close the current program, if it is running or paused.



Pause a running program at its current location.



Step to the next statement, this will step into a subroutine if the next statement calls a subroutine. You can also use this to start and pause at the first statement in a program.



Step over the next statement. This is the same as the step unless the statement is a subroutine, in which case the entire subroutine is run and the program is paused when the subroutine has completed.



Step out will complete the current subroutine and pause when the subroutine is exited. Therefore this method only really applies when you are currently paused within a subroutine.

Break Points



Toggle setting a break-point on the current line. The program will break and pause when a break-point line is reached. Break-points are shown as red circles on the left of the code window. Break-points can also be set or unset by clicking to the left of the line numbers. A red circle appears in the first column of the left margin when a break-point is set.



Remove all break-points.



Toggle to temporarily disable all break-points.

Book Marks

Bookmarks do not interact with the debugger, but can be useful in a long program to quickly move between different sections of the program.

Watch Variables

Variables may be added to the watch list in the Debug tab at the bottom of the IDE. The variable name may be entered directly. Alternatively a variable may be selected in the program code and added with Ctrl+W or using right click context menu.

Whenever the program is paused the current value of the watch variables will be shown.

Various options for the watch variable may be used to change the current value of a variable, or pause the program when the value of the variable changes or becomes greater than or less than or equal to a set value.

You can also use the [] notation to watch array elements, e.g. data[5] or data[i].

Variables and Call Stack



Use this option to create a window that displays all of the program variable values and subroutine call stack every time the program is paused.


Example

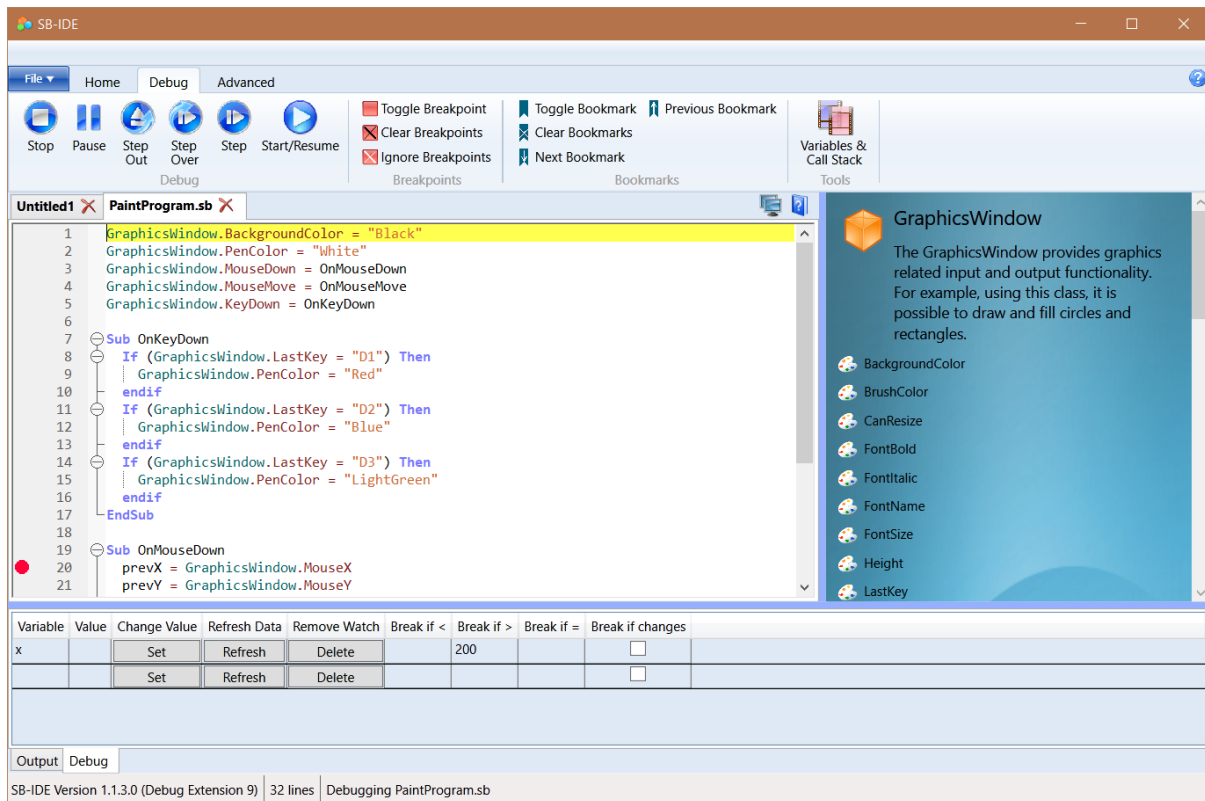
Using the *PainProgram.sb* example we are going to set a break-point inside the *OnMouseDown* event subroutine and set a watch parameter to pause the program when the variable 'x' is greater than 200.

Doing this should leave your program looking like this:

The screenshot shows the SB-IDE interface with the *PaintProgram.sb* file open. A breakpoint is set at line 19, which is the start of the *OnMouseDown* subroutine. The watch window at the bottom shows the variable *x* with a value of 200. The *GraphicsWindow* class is also visible on the right side of the interface.

Variable	Value	Change Value	Refresh Data	Remove Watch	Break if <	Break if >	Break if =	Break if changes
x		Set	Refresh	Delete		200		<input type="checkbox"/>
		Set	Refresh	Delete				<input type="checkbox"/>

Now we can either run the program or step into the first line – we do the second by pressing Step .



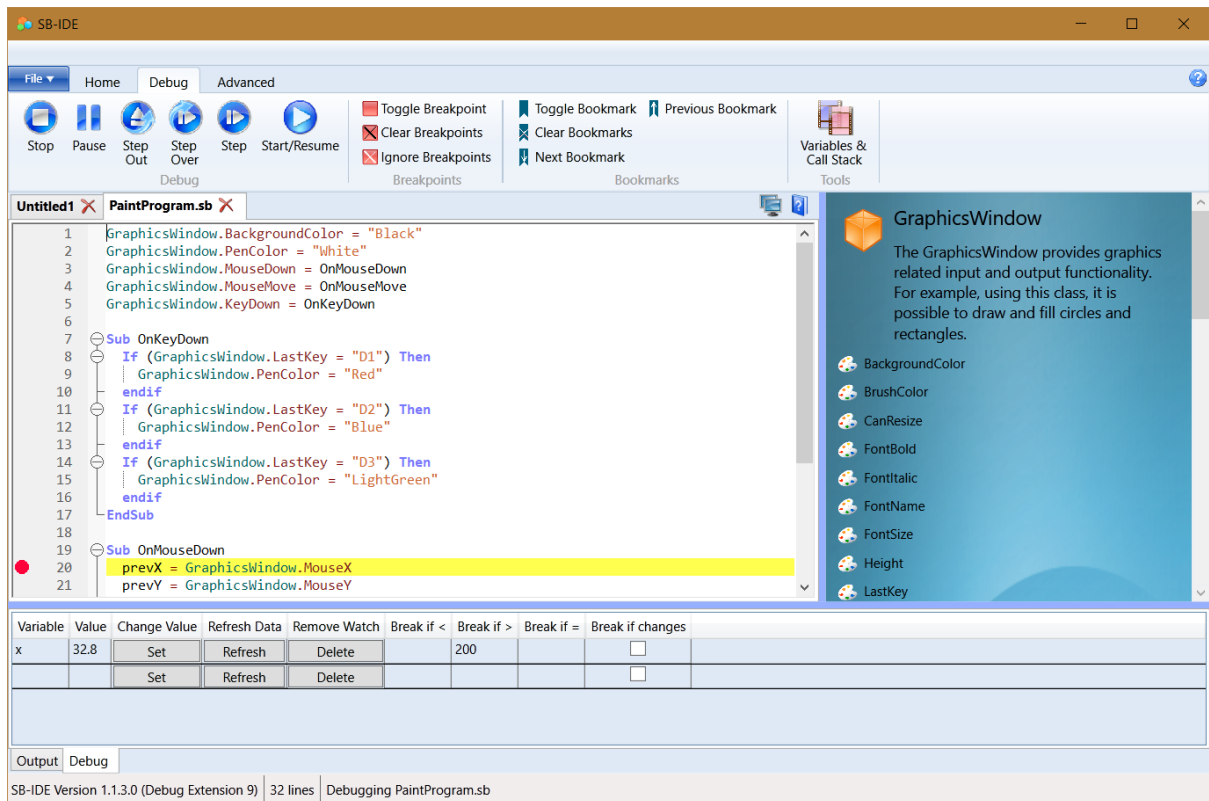
The program is now paused at the first line, the variable x has not been set yet so no value is shown. Note that default values for unset variables in Small Basic is zero, so this could be confusing if you set conditional watch break-points that trigger for a value of zero (e.g. $x < 200$).

Now we can continue by pressing Start/Resume .

The program is now running and the GraphicsWindow with a black background appears. We can move our windows or select the GraphicsWindow to interact with it.



Draw a line in the left part of the window, $x < 200$. Hint, make sure you enter the GraphicsWindow from the left, otherwise x may be > 200 .

As soon as you press the mouse down the program pauses at the break point.



Note that the variable 'x' now has a value.

The current value of any variable can also be view when paused by hovering the mouse over a variable name in the program.


You can also use the Variables and Call Stack button  to bring up a window that shows the values of all variables. Advance one step  to update the data in this window.

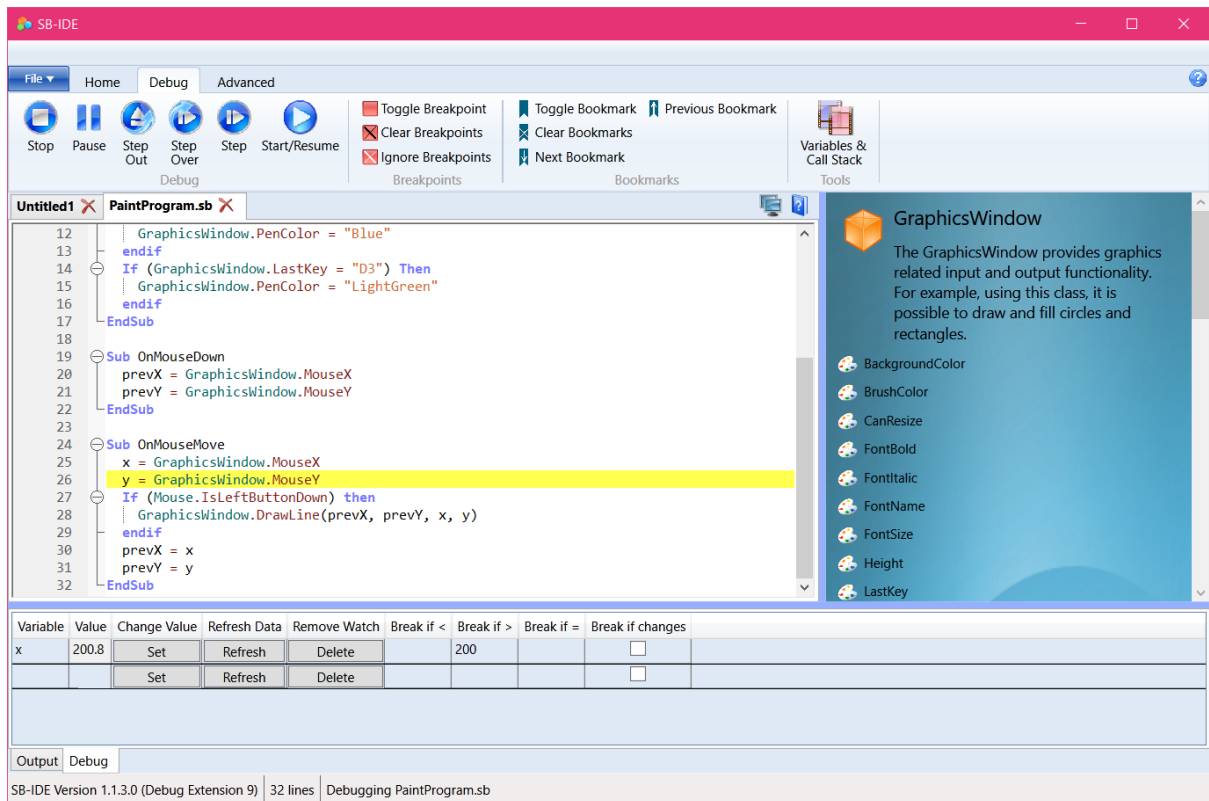
Variables & Call Stack

Variable	Value
prevx	32.8
prevy	76
x	32.8
y	76

Call Stack
onmousedown

Now remove the break point in the subroutine *OnMouseDown*, simply by clicking the red circle in the left margin on line 20.

Press  to continue the program, and draw a line, this time that has $x > 200$. At any point when x become greater than 200 the program will pause.



You can experiment changing the conditions for watch variables to pause the code. You can also change the value of a variable by typing a new value in the Value column of the watch list and pressing the Set button.

This example is useful to show how to debug, but there is an issue with some aspects of this program under some circumstances. The problem is that the main UI thread actually ends and all the action occurs inside event subroutines. For programs of this type it is sometimes useful to add a dummy 'Game Loop' to keep the main UI thread running, perhaps like this:

