

# Introduction

This tutorial introduces the following SingleStep features:

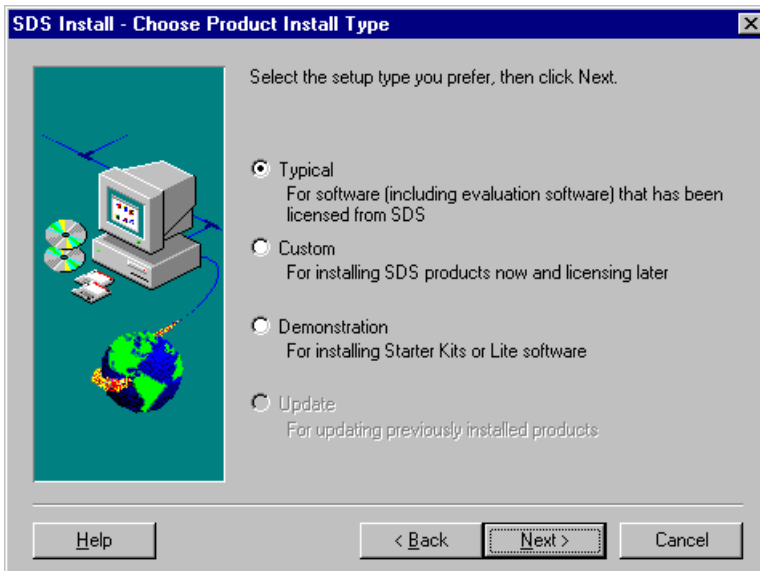
- Starting a debug session.
- Watching variables.
- Setting breakpoints.
- Modifying breakpoints.
- Stepping through a program.
- Changing variables as the program runs.
- Using the command line.
- Reading variables.

This tutorial utilizes the *SingleStep Demonstration Software - Starter Kits* for the 68K and PowerPC processors. You can also follow this tutorial with Evaluation or Licensed Software. (To follow this tutorial for the M•CORE processor, use the provided M•CORE example program files with Evaluation or Licensed Software. See the *Installation and Notes* pamphlet and Chapter 1.1 of the *SingleStep Users Guide* for installation information.)

# Installation

To install the *SingleStep Demonstration Software (Starter Kit)*:

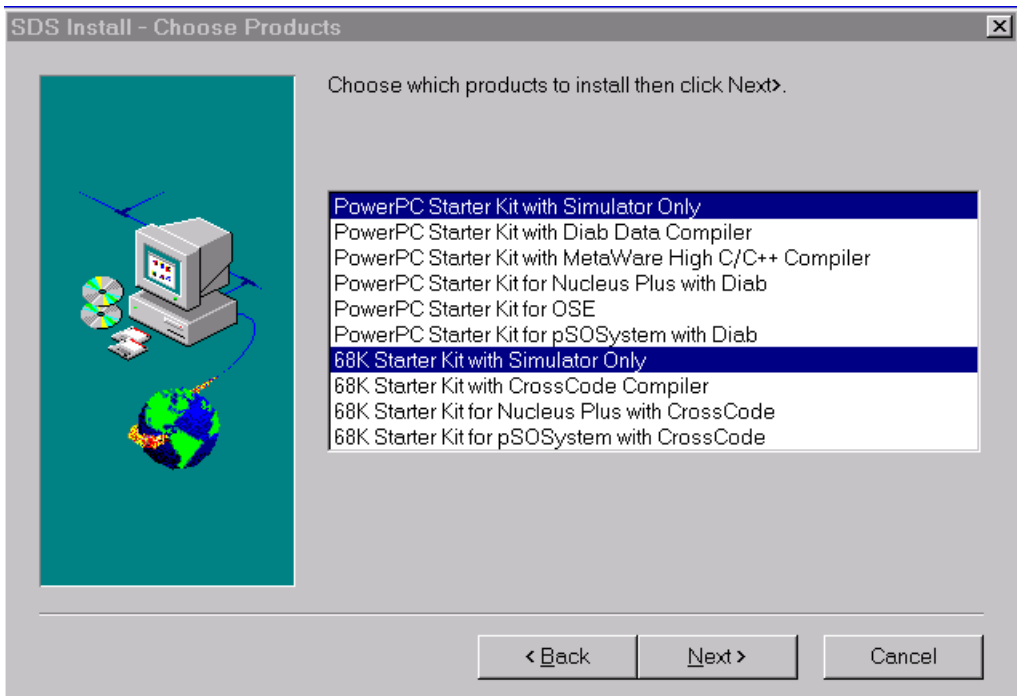
1. Run *setup.exe* from the CD-ROM.
2. Select *Demonstration (For installing Starter Kits or Lite software)* on the *Choose Product Install Type* page, then click on the *Next>* button.



On the next screen, select *Starter Kits* and click *Next>*.

A wide variety of Starter Kits are available for 68K and PowerPC processors.

- 3.** Select *PowerPC Starter Kit with Simulator Only* and *68K Starter Kit with Simulator Only*, then click on the *Next>* button.



- 4.** On the remaining installation dialogs, click on the *Next>* button to choose the default selections.

# SDS Products

The SingleStep debugging environment comes in a wide variety of configurations. Different versions of SingleStep may be used to develop embedded systems based on 68K, PowerPC, M•CORE or ColdFire processor families. SingleStep 7.4 runs on Windows 95 and 98, Windows NT 3.5.1 and 4.0, Sun Solaris 2.5 and later, and HP-UX 10.20 host platforms. (If support for Windows 3.1x, SunOS, or earlier versions of Sun Solaris or HP-UX is required, SingleStep 7.11 is still available for these platforms.)

SingleStep may be purchased in the following configurations:

**Simulators.** Available for 68K, PowerPC, and M•CORE. Simulators reproduce the behavior of real or theoretical target hardware on your host platform. This allows you to debug software before hardware is available or when the target hardware is not connected to your host. The SingleStep Demonstration Software (Starter Kit) is a simulator with some features disabled. (The simulator supplied with the CrossCode C/C++ compiler also has some features disabled.)

**Target Monitors.** Available for 68K, PowerPC, M•CORE, and ColdFire. A target monitor is a small program that runs on your target hardware. The monitor provides communication between the target and host computer through a serial connection, a parallel connection, or a ROM emulator.

**On Chip Connections.** Available for 68K, PowerPC, and ColdFire. Chips with BDM (Background Debug Mode), JTAG, or OnCE have built-in mechanisms which give an external debugger access to and control over to information on the CPU. Since the communication software is built into the CPU, these systems consume fewer target resources than target monitors. These processors communicate to the host through small, inexpensive connectors.

**Emulators.** Available for 68K, PowerPC, and ColdFire. Emulator debugging hardware is connected between your host computer and your target hardware. Emulators govern the communication between your host and target, so you can debug without consuming target resources.

**Kernel Awareness.** Available for a wide variety of Real Time Operating Systems (RTOSes). RTOS awareness allows SingleStep to see and interact with all tasks running in the target system. Some RTOS packages also support Task Debug Mode, which allows you to stop one task while others continue to run. You can develop task awareness for unsupported or custom RTOSes using the Multi-Task Debugging Adaption Kit.

# Getting Started

## 1. Start either *SingleStep Demo (68K)* or *SingleStep Demo (PowerPC)*.

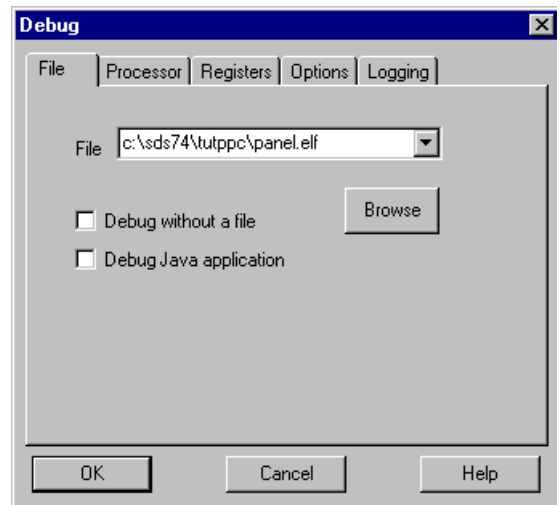
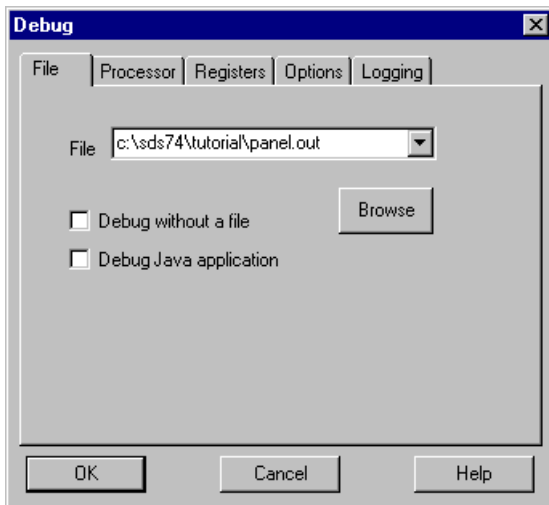
- ◆ On Windows 95, 98, and NT 4.x, choose a SingleStep product from the *SingleStep 7.4* submenu of the *Programs* submenu of the *Start* menu.
- ◆ On Windows NT 3.5.x double-click the SingleStep product icon in the *SingleStep 7.4* program group in the *Program Manager*.

When you first open SingleStep, the *Demo License Dialog* appears; then when you click *Continue*, the *Debug Dialog* opens.

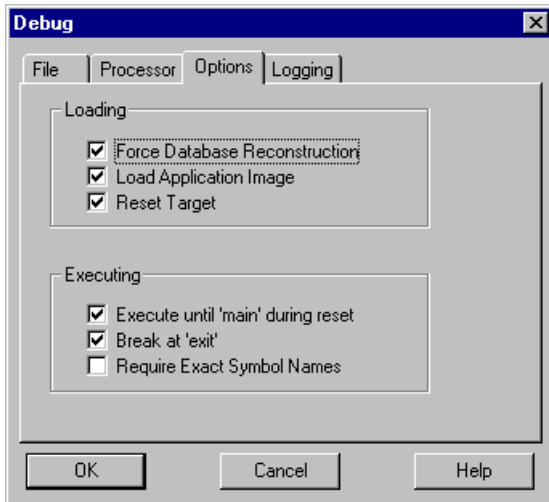
## 2. Click the *Browse* button to choose an object file to debug. The 68K versions of SingleStep can read *.out* files produced by the CrossCode C/C++ compiler, IEEE-695 files (IEEE-695 is only supported for C, not C++), or *.elf* files produced by other compilers. The PowerPC versions of SingleStep read only *.elf* files.

If you are using a 68K version, choose the file  
C:\sds74\tutorial\panel.out

If you are using a PowerPC version,  
choose the file  
C:\sds74\tutppc\panel.elf

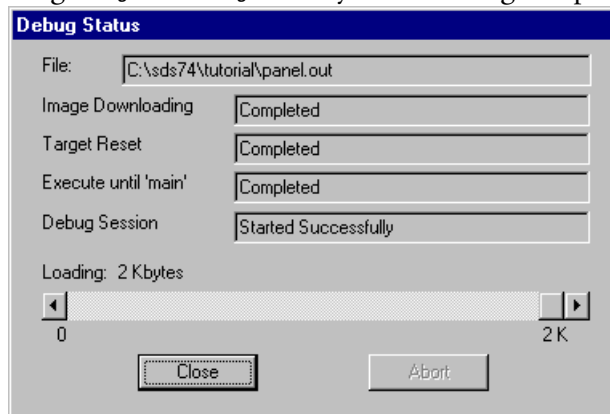


3. Click the *Options* tab. Make sure that all options except *Require Exact Symbol Names* are selected.

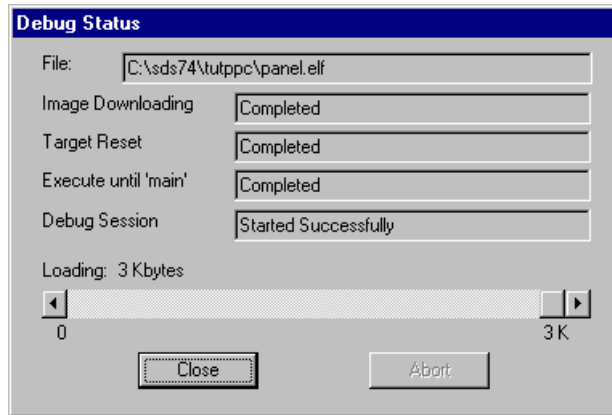


If you would like to know what any of these controls do, click the *Help* button. This takes you to the on-line help for the *Debug Dialog*. Click the *Options* tab in the help window, then click in the help window on the control you want to know more about.

4. Click the *OK* button in the *Debug Dialog*. SingleStep downloads the object file to the target. In this case, the target is the Simulator. With other versions the target may be actual hardware. SingleStep then runs the program on the target up to the first break-point. The resulting *Debug Status Dialog* shows you what SingleStep did.



68K

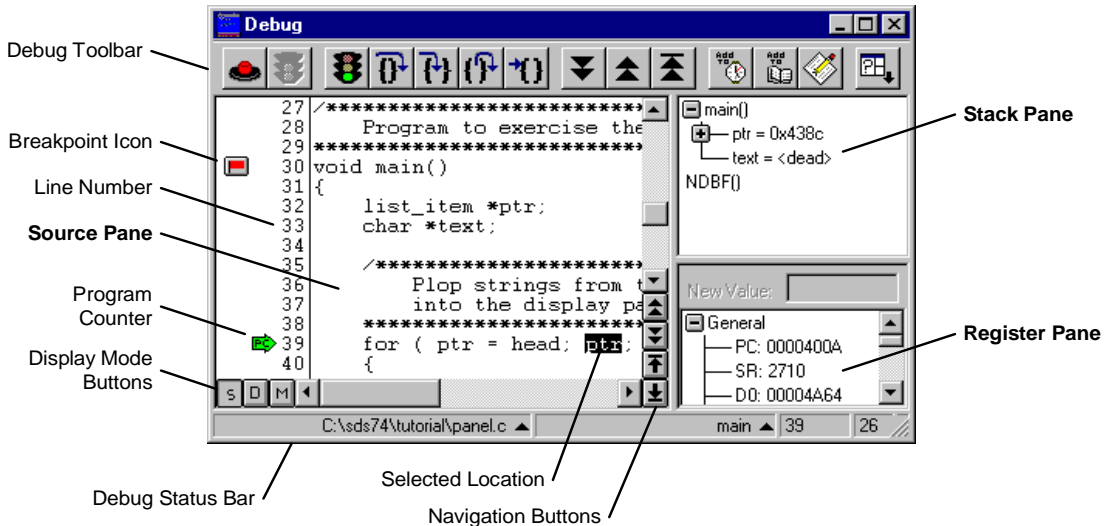


PowerPC

5. Click the *Close* button.

# The SingleStep Debug Window


When you first start SingleStep, the *Debug Window* and the *Watch Window* open. (The *Watch Window* is hidden behind the *Debug Window*.) Most of your basic debugging information will come from these windows. The *Debug Window* uses three areas, called *Panes*, to show different information about your object file. The large *Pane* with code in it is called the *Source Pane*. The smaller area with functions in it is called the *Stack Pane*. The area with register IDs and values is called the *Register Pane*. The picture below shows the *Debug Window*. The *Watch Window* has a single area where the values of the variables are displayed. You will see how it works as you explore the *panel* program.



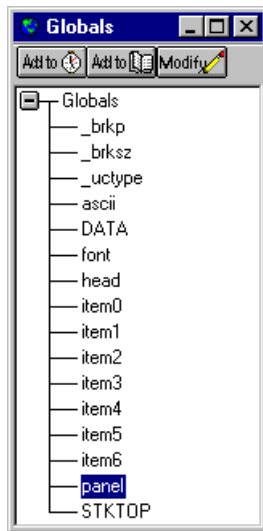


# The Panel Program

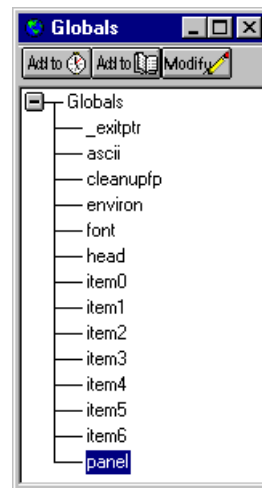
The program you downloaded to your target (*panel.out* or *panel.elf*) simulates a simple text read-out on an LED display. The display is assumed to be an array of dots or pixels that can be turned on and off by setting the proper bits in memory. A variable called *panel* simulates the LED display with an array of ASCII characters. This variable represents lit pixels with the character "#" and represents unlit pixels with a space.


1. Click the  (*Globals*) button in the toolbar. This opens the *Globals Window*.
2. Select the *panel* variable.

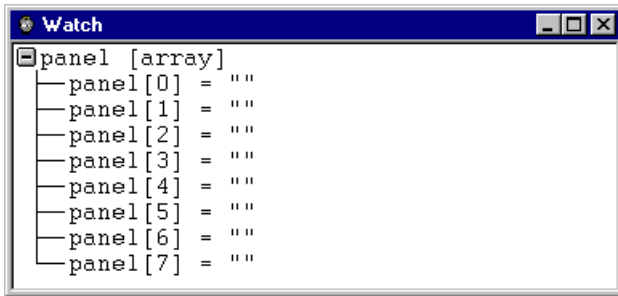
68K



PowerPC



- Click the  (Add to Watch) button in the *Globals Window*. The *Watch Window* displays the structure and content of *panel* as shown below.

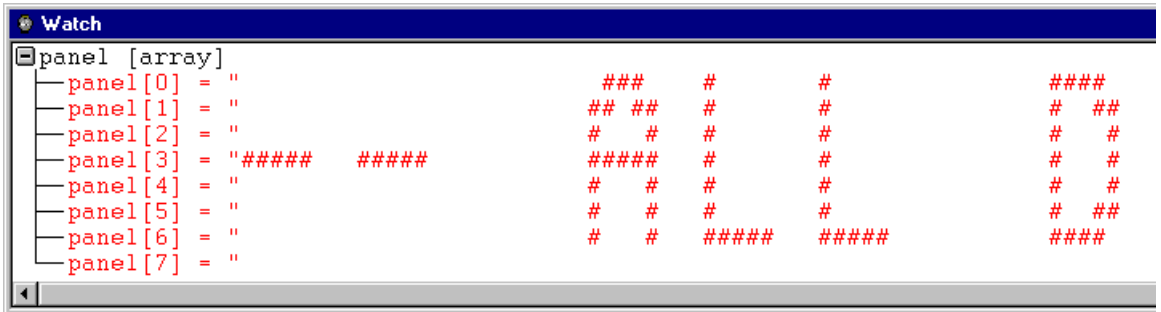


*panel* is a 2 dimensional array of *char* defined as

```
char panel[HEIGHT][WIDTH*PANELSZ];
```


where *HEIGHT* is 8 and each row is a *NULL*-terminated array of *chars* (that is, each row is a string).

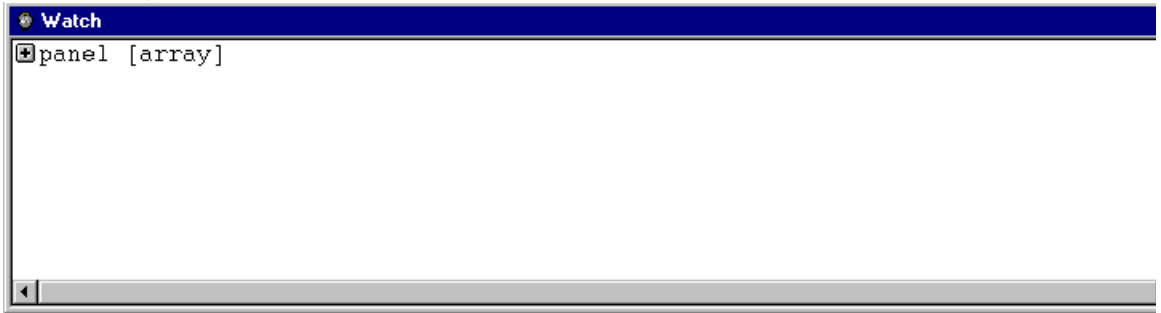
- Click the  (Go) button. The *Watch Window* should look something like the picture below.



It now shows the value of *panel* when the program stopped (at the end of the program). To see *panel* as it changes we need to set some breakpoints.

**Note:** variables change color to indicate a change in value since the prior SingleStep operation. The words "-- ALL DONE!" will be red.

5. To hide the individual lines of *panel*, click on the  (*Close Expansion*) button next to *panel*. The *Watch Window* then shows that *panel* is an array, but does not show what is in the array.



6. To view the lines in *panel* again, click the  (*Expand*) button.


The *Watch Window* displays any structured variable (*array*, *struct*, or *pin*) in this manner. Simple variables (*int*, *float*, *char*, etc.) are displayed as a single line.

# Seeing an Embedded Program Run


When you ran the *panel* program, you knew something happened because the *panel* variable was shown in the *Watch Window*. If the program were designed to work with an actual LED display, and you had the proper hardware with the LED display properly hooked up, you also would have seen text moving across the display. Typically when you work on an embedded program, the hardware does not provide much feedback as the program runs (because the hardware is in development, or because it does not have a display). This (among other reasons) is why you need SingleStep.


To see how SingleStep works, we're going to use it to understand how the *panel* program works.

If the *Globals Window* is still open, select the window, then close it by typing the key combination *Ctrl-F4* (hold down the *Ctrl* key and press the *F4* key).

1. Click the  (*Reset*) button, then click the *OK* button on the confirmation dialog. The line

```
    for ( ptr = head; ptr; ptr = ptr->next )
```

should be visible in the *Debug Window*. (If it is not, you might be displaying the program in *Mixed Mode* or *Disassembly Mode* rather than *Source Mode*. Select *Source Mode* by clicking on the  (*Source Mode*) button at the bottom of the *Source Pane*.)

2. Click the  (*Close Expansion*) button next to *panel* in the *Watch Window*.
3. *ptr* in this *for* statement is a linked list defined elsewhere as:


```
typedef struct list
{
    struct list *next;
    char text[20];
} list_item;



list_item *ptr;
```

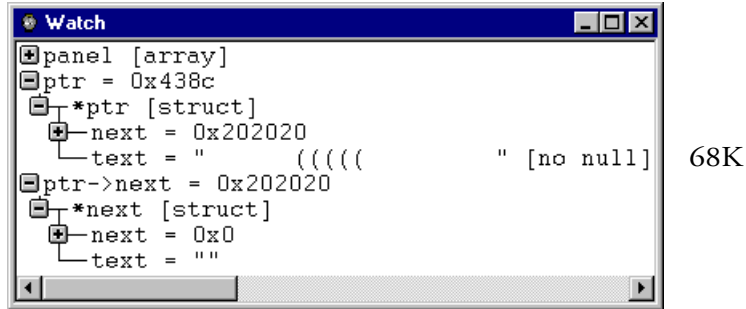
Double-click on *ptr* in the expression *ptr->next* on the line

```
    for ( ptr = head; ptr; ptr = ptr->next )
```

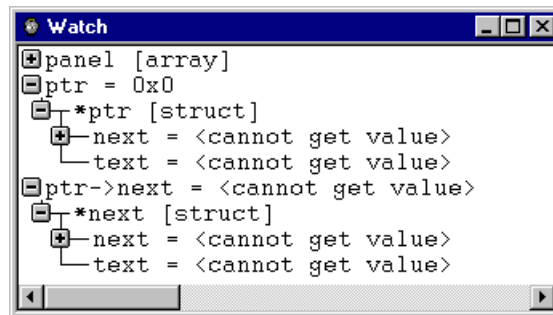
This highlights *ptr*.

4. Click the  (*Add to Watch*) button in the toolbar at the top of the *Debug Window*. This adds the *struct* pointer *ptr* to the *Watch Window*.

5. Double-click on *next* in the variable *ptr->next*. This highlights *ptr->next*.
6. Click the  (*Add to Watch*) button. This adds the *struct* element *ptr->next* to the *Watch Window*.
7. Expand *ptr* and *ptr->next* one level by clicking the  (*Expand*) buttons. Neither of these variables are initialized yet, so the display will be different on the two demonstration simulators, as shown below.




68K




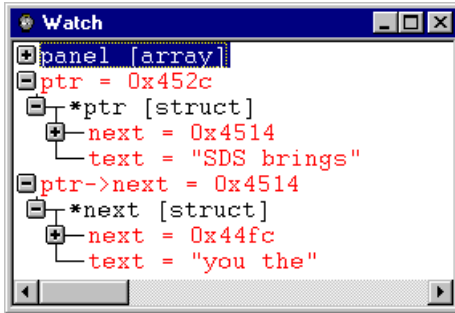
PowerPC

8. In the *Source Pane*, double-click on the line number next to
 

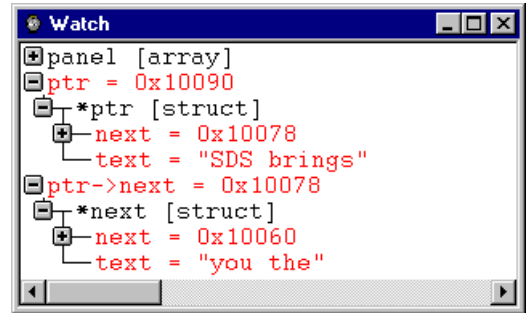
```
display_str( 0, ptr->text, 1 );
```

 This inserts a  (*Breakpoint*) icon next to the line number.

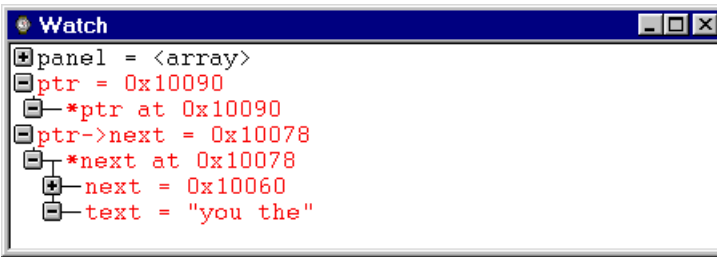
9. Click the  (Go) button. Notice the changes in *ptr* and *ptr->next*.




68K





PowerPC

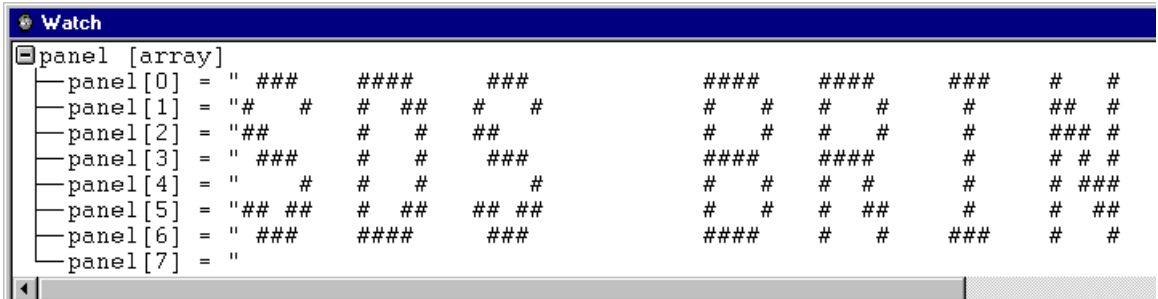


*ptr->text* now has the value "SDS brings" or 0x10078 and *ptr->next->text* now has the value "you the".



Also notice the  (Program Counter) icon in the *Source Window* moved from the initial stop location to the current breakpoint.

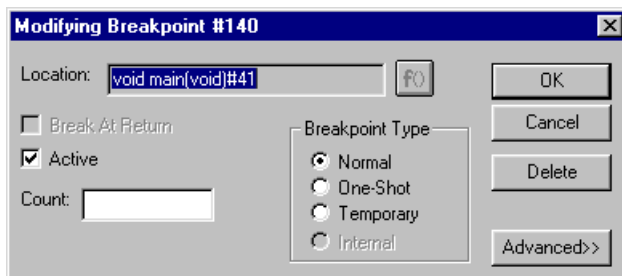
10. Click the  (Go) button again. You can see *ptr->next* has been copied to *ptr*, and *ptr->next* now has new values.

- 11.** Click the  (*Expand*) button next to *panel*. You can now see the text which was previously in *ptr->text* displayed on the panel.



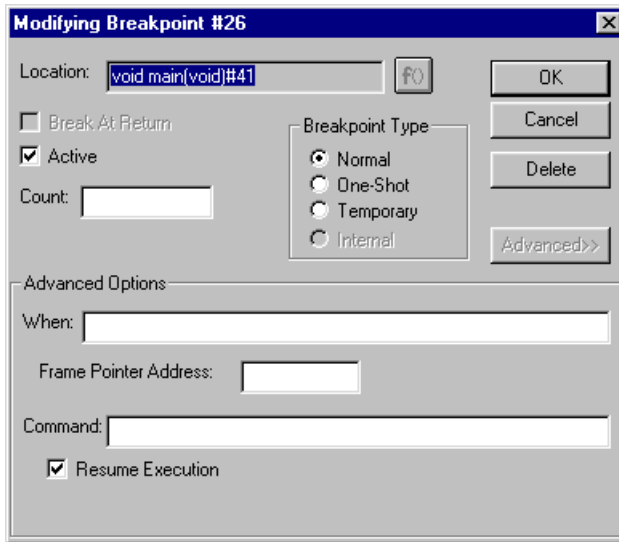
```
Watch
panel [array]
- panel[0] = " ###      ####      ###      ####      ####      ###      #      #
- panel[1] = "#      #      #      #      #      #      #      ##      #
- panel[2] = "##      #      #      ##      #      #      #      ###      #
- panel[3] = " ###      #      #      ###      ####      ####      #      #      #
- panel[4] = "      #      #      #      #      #      #      #      ###      #
- panel[5] = "##      ##      #      ##      ##      ##      #      #      ##
- panel[6] = " ###      ####      ###      ####      #      #      ###      #      #
- panel[7] = "
```

- 12.** Click the  (*Breakpoint*) icon next to the  (*Program Counter*) icon. This opens the *Modifying Breakpoint Dialog*.



- 13.** Click the *Advanced>>* button. This expands the dialog box.

**14.**Select the *Resume Execution* checkbox.





**15.**Click the *OK* button.

**16.**In the *Source Pane*, scroll down a few lines so the line

```
panel_rotate( -1 );
```

is visible. Insert another breakpoint by double-clicking on the line number next to this line.







**17.**Open the *Modifying Breakpoint Dialog* for this new breakpoint, make the same change you made on the other breakpoint, and click *OK*.


**18.**Click the  (*Reset*) button, then click the  (*Go*) button. Now you can see the entire *panel* program as it runs. The display in the *Watch Window* will simulate the actual LED display on the real hardware.

## Looking at the Details



You have now seen what the *panel* program does, but you have not yet seen how it works. To do this we are going to step through the program.




1. Click the  (*Reset*) button. This sets the  (*Program Counter*) icon back to the start of *main*.
2. Click the  (*Step into*) button. This runs one statement moving the  (*Program Counter*) icon to the next executable statement. On the 68K simulator, this runs the assignment `ptr = ptr->next`. The next executable statement is the test (on the same line) in the *for* statement. On the PowerPC simulator, the next executable statement is the *display\_str* function.
3. If you are running the 68K simulator, click the  (*Step into*) button again. The  (*Program Counter*) icon moves to the *display\_str* function call.




If you are running the PowerPC simulator the  (*Program Counter*) icon already points to the *display\_str* function call.

Notice the status bar at the bottom of the *Debug Window*. The first item indicates the source file. Up until this point, you have been looking at the file *panel.c*.




4. Click the  (*Step into*) button again. This time the  (*Program Counter*) moves to the first statement of the *display\_str* function. *display\_str* is defined in the file *panel2.c*, so the *Source Pane* now displays this file.





You can look at *panel.c* again (or look at any other file linked into the object you are debugging), by clicking on the  button next to the file name and choosing a file from the list.

5. Continue clicking the  (*Step into*) button. As you go, read the comments above the  (*Program Counter*) icon position to figure out what the statements do. Stop after about 10 to 15 clicks, when the  (*Program Counter*) icon is on the line:

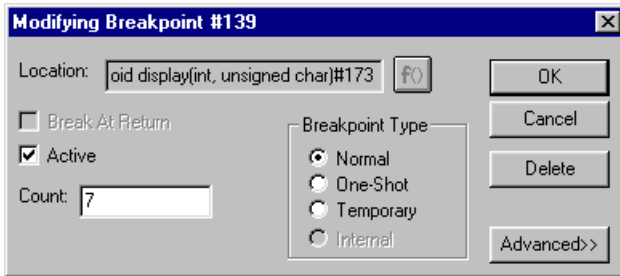
```
*panelp++ = (bitmap & mask) ? ON: OFF;
```

6. Click the  (*Step into*) button until *panel[0]* changes from "" to " #". If the  (*Program Counter*) icon has not returned to the line with *\*panelp++*, continue clicking the  (*Step into*) button until it does.

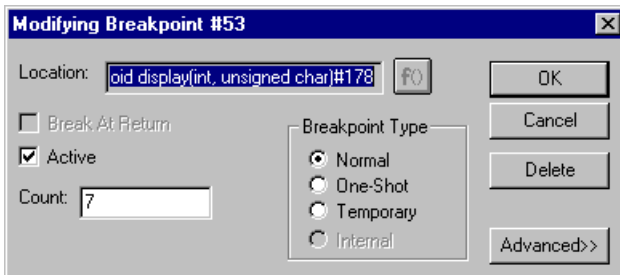
**Note:** If you accidentally go too far, set a breakpoint next to the line with *\*panelp++*, reset and run the program, then click the  (*Step into*) button until you are at the right place.

7. Add a breakpoint to the current line, and open the *Modifying Breakpoint Dialog* by clicking on the  (*Breakpoint*) icon.




8. Specify a *Count* of 7, and click *OK*.



68K



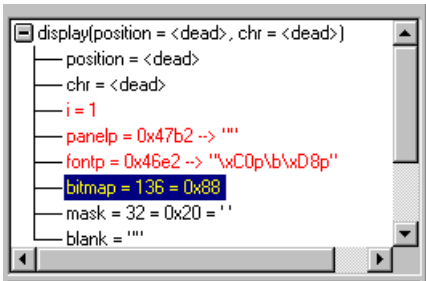
PowerPC

9. When you click the  (*Go*) button you will see the program build the first row of the first character on the panel. The program stops when *panel[1]* is "#". (If *panel[1]* is "" you have not gone far enough. Click the  (*Step into*) button until *panel[1]* changes and the  (*Program Counter*) icon returns to the line with *\*panelp++*.)

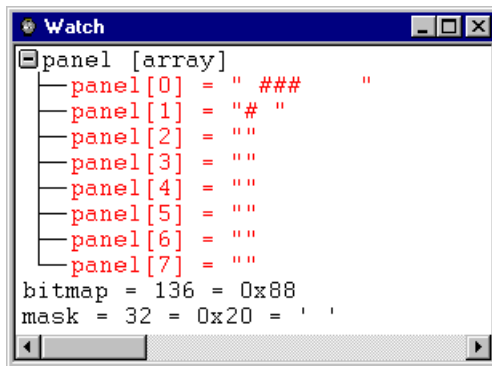
How does the *panel* program do this? Just watching the program execute may not make this obvious. To show what the program is doing, try the following:

1. Make the *Watch Window* a little taller.
2. Select *ptr* and press the right mouse button while the mouse is over the *Watch Window*. This opens a pop-up menu.
3. Choose *Remove* from the pop-up menu. This removes *ptr* from the *Watch Window*.
4. Remove *ptr->next* from the *Watch Window* the same way.

**5.** Select *bitmap* in the *Stack Pane* of the *Debug Window*.



Press the right mouse button and select *Add 'bitmap' to Watch window* from the popup menu. This adds *bitmap* to the *Watch Window*. Add *mask* to the *Watch Window* the same way. The *Watch Window* should now appear as shown below:



**6.** Step through the program until *panel[2]* changes, observing what happens as you go.

Notice that *mask* starts with a value of 0x40, which has a binary value of 01000000. *bitmap* is a binary representation of one row of one character. Each step shifts the bit in *mask* one column to the right. That is, as the program constructs a *panel* row, *mask* changes from 0x40 to 0x20 (binary 00100000), then changes to 0x10 (binary 00010000), and so on down to 0x00 (binary 00000000). Meanwhile, *bitmap* stays unchanged until the beginning of the next row.

Just before the first column in the next row, *mask* resets to 0x80 (binary 10000000), and the sequence starts again with a new *bitmap*.

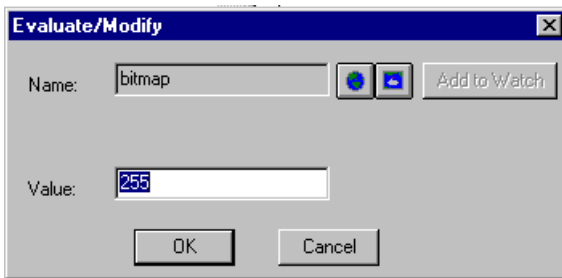
The 1 bit in *mask* thus acts as a pointer to the bit in *bitmap* that will be written to *panel* next.

# Variables and Commands


To see what is going on a little more clearly, we are going to modify a program variable (*bitmap*), then set a breakpoint that triggers when (*panel[3]*) changes.

**Note:** This example illustrates the SingleStep scripting language by showing you how to set a breakpoint that will trigger when a variable changes. It is a complex example compared to the rest of this tutorial, and may look intimidating at first. The scripting language is designed to be a powerful debugging tool that is easy to use if you are familiar with programming languages. The *Command Window* is actually an implementation of a UNIX-style C Shell, so if you are familiar with this type of UNIX shell you will be familiar with the command language syntax.

1. Select *bitmap* in the *Watch Window* and choose *Modify* from the pop-up menu. This opens the *Evaluate/Modify* dialog.
2. Change *Value* from *192* (binary 11000000) to *255* (binary 11111111), then click the *OK* button.



Now we are going to run the program until *panel[3]* changes. The last time we set a breakpoint like this, we guessed that we needed to go past the breakpoint 7 times before *panel[2]* would change. This time we are going to explicitly specify that the breakpoint only triggers when *panel[3]* changes. The following shows you just a little of the debugging power accessible from the SingleStep command line.

1. Click the  (Command) button. This opens the *Command Window*. When you open the *Command Window*, you may not have a prompt. This is OK. You can type commands anyway, or you can press *Enter* to get a prompt. An empty command line with a prompt looks like the line below.

```
SingleStep>
```


2. Type the following in the *Command Window* and press *Enter*.

```
set oldpanel = "`read -rY panel[3]`"
```

This sets the debugger variable *oldpanel* to the current value of the program variable *panel[3]*:

- ◆ The backticks (``) tell SingleStep to evaluate the *read* statement before assigning a value to the debugger variable.
- ◆ The double-quote protects the value returned by the *read* statement.
- ◆ The *-r* option tells SingleStep that the value of *panel[3]* might have changed since the last time it was read. This forces SingleStep to reread the actual value.
- ◆ The *-Y* option tells SingleStep to expand structures or arrays, thus reading *panel[3]* as a string.

**Note:** The SingleStep scripting language is case-sensitive. The *-r* option is different from the *-R* option.

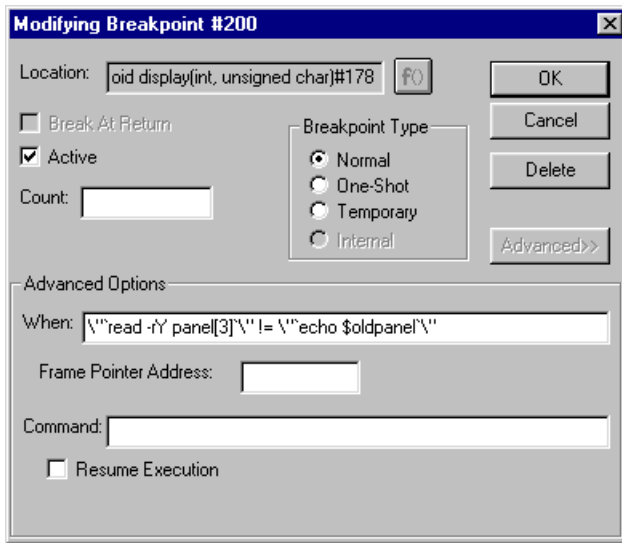
3. Now, in the *Source Pane* click on the  (Breakpoint) icon next to:

```
*panelp++ = (bitmap & mask) ? ON: OFF;
```

4. In the *Modifying Breakpoint Dialog*, click on the *Advanced>>* button, then type the following in the *When* field under *Advanced Options*:

```
\``read -rY panel[3]``" != \``echo $oldpanel``"
```

**Note:** Make sure the spaces, double-quotes, and backticks (``) are exactly as they appear above; otherwise the parser may not interpret the statement correctly.





Click on *OK*. The next time the program is run, it will stop when *panel[3]* changes.

This *When* statement is more complicated than most; however, with a little bit of thought it is easy to decipher:

- ◆ There are two statements (a *read* and an *echo*) on either side of a conditional operator (*!=*). Since the two statements are enclosed in backticks (`), they are evaluated before the not-equal condition (*!=*).
- ◆ The *read* statement returns the current value of *panel[3]*.
- ◆ The *echo* statement returns the value of the debugger variable *oldpanel* (which was set earlier).
- ◆ Both statements return strings which have to be enclosed in double-quotes (") to work with conditional operators.
- ◆ Since the statement is being typed in a dialog box, not on the command line, the double-quotes need to be preceded with the backslash character (\).

When the current value of *panel[3]* is not equal to the saved value of *panel[3]*, the breakpoint will trigger.

5. Now we are ready to see what happens when we click the  (*Go*) button. Remember, we changed *bitmap* so its value is 0xFF (binary 11111111). Click  (*Go*). The program stops when *panel[3]* is " ". *panel[2]* is now "#####".


Since setting this up was a little complicated we are going to change *bitmap* again, and use similar commands to stop the program when *panel[4]* changes.

1. Change the value of *bitmap* from 112 (binary 01110000) to 7 (binary 00000111). (Make sure there are no quotes around the 7.)
2. In the *Command Window* press the up-arrow key until the *set* command you typed earlier appears.
3. Change *panel[3]* in the command to *panel[4]*, then press the *Enter* key.

```
set oldpanel=`read -rY panel[4]`
```


4. Change *panel[3]* in the *When* field of the breakpoint to *panel[4]*, and click the *OK* button in the *Modifying Breakpoint Dialog*.

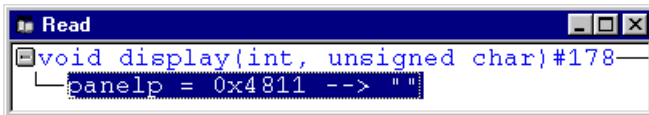
```
\``read -rY panel[4]``" != \``echo $oldpanel``"
```

5. Click the  (Go) button. The program runs until *panel[4]* changes. Since we changed *bitmap* to 7 (binary 00000111), *panel[3]* is now " ###" (five spaces and three # characters).

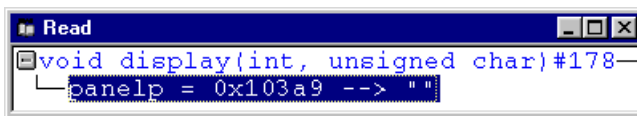
## The Read Window

By now it should be fairly clear what *bitmap* and *panel[]* do. What does *panelp* do? To look at this variable, we are going to use the *Read Window*.


1. Change the *When* condition in the breakpoint to stop when *panel[2]* changes. Reset the program, then set *oldpanel* to the current value of *panel[2]*. Now run the program.
2. In the *Source Pane*, double-click on *panelp*, then click on the  (Add to Read) button. This opens the *Read Window* with *panelp* displayed similarly to variables in the *Watch Window*. Notice the address of *panelp* is different on the 68K and PowerPC.



68k



PowerPC

3. Now also add *panelp* to the *Watch Window*.
4. Run the program until *panel[2]* changes again (set *oldpanel* to the current value of *panel[2]*, then click the  (Go) button). The current value of *mask* is 0x20 (binary 00100000), and the current value of *bitmap* is 0xC0 (binary 11000000). Notice that the value shown for *panelp* in the *Watch Window* has changed, but the value in the *Read Window* has not. The *Read Window* is used to remember previous values of variables, while the *Watch Window* is used to show current values of variables.
5. Change *panelp* back to the old value (0x4811 for 68K, or 0x10701 for PowerPC), then run the program until *panel[2]* changes again. The # in the second column changed to a space. Why did this happen?

The value of *panelp* controls where in *panel[]* the next character is written. By changing *panelp*, we forced the character that would normally be written to the third position in the array to be written to the second position instead.
6. Continue stepping through the program until *panel[3]* changes. Notice that *panel[2]* is now missing a character (because we wrote both the second and third characters to the second position in the array).

This concludes the Starter Kit Tutorial. You should now understand how the demonstration program *panel* works and how to use the fundamental features of SingleStep. Several important features were not covered here. These are covered in the main SingleStep manuals and the on-line help.



# Index

## Symbols

.elf files 5

.out files 5

## Numerics

68k support 4

## A

Add to Read button 23

Add to Watch button 10, 12

## B

background debug mode 4

BDM 4

Breakpoint icon 13, 15, 17, 21

buttons

Add to Read 23

Add to Watch 10, 12

Close Expansion 11

Command 21

Expand 11

Globals 9

Go 10, 14, 17, 18, 22, 23

Help 6

Reset 12, 17

Step into 17

## C

chips supported 4

Close Expansion button 11

ColdFire support 4

Command button 21

Command Window 21, 23

connections 4

## D

Debug Dialog 6

Debug Options 6

Debug Status Dialog 6

Debug Window 8, 19

illustration 8

status bar 17

Demo License Dialog 5

dialogs

Debug 5, 6

Debug Options 6

Debug Status 6

Demo License 5

Evaluate/Modify 20

Modifying Breakpoint 15, 17, 21

display modes 12

## E

elf files 5

emulators 4

Evaluate/Modify Dialog 20

Expand button 11

## G

getting help 6

getting started 5

global variables, listing 9

Globals Window 12

Go button 10, 14, 17, 18, 22, 23

## H

Help button 6

## I

icons

Breakpoint 13, 15, 17, 21

Program Counter 14, 15, 17

IEEE-69x files 5

## J

JTAG 4

## K

kernel awareness 4

## **M**

M•CORE support 4

Modifying Breakpoint Dialog 15, 17, 21

Multi-Task Debugging 4

## **O**

object files 6

- panel programs 9

- supported formats 5

on chip connections 4

OnCE 4

on-line help 6

operating systems

- real time 4

- supported 4

out files 5

## **P**

panel program, description 9

panes

- Register 8

- Source 8

PowerPC support 4

processors supported 4

Program Counter icon 14, 15, 17

## **R**

read statement 21

Read Window 23, 24

real time operating systems 4

Register Pane 8

Reset button 12, 17

RTOS 4

## **S**

scripting language 20

simulators 4

Source Mode 12

Source Pane 8, 21

Stack Pane 19

status bar 17

Step into button 17

## **T**

target monitors 4

Task Debug Mode 4

## **W**

Watch Window 10, 13, 15, 23

- adding variables to 10, 12

- removing variables from 18

windows

- Command 21, 23

- Debug 8

  - illustration 8, 19

  - status bar 17

- Globals 9, 12

- panes 8

- Read 23, 24

- Watch 8, 23

  - adding variables to 10, 12

  - illustration 10, 15

    - changes displayed 13

  - removing variables from 18

---

**Manual:** ©1982-1998 SDS. All Rights Reserved

**Software:** © 1982-1998 SDS. All Rights Reserved

Printing Revision: 100003-11

A notice of protection under copyright, trade secret and contract law may be printed from each tool via use of the -C option on the tool's command line.

The use and copying of the Users Manual and Software are subject to the License Agreement packaged with this manual and/or software. Any other use is prohibited.

---

CrossCode® is a trademark of SDS.

SingleStep™ is a trademark of SDS.

M•CORE is a trademark of Mototola.

Windows® is a registered trademark of Microsoft.

UNIX® is a registered trademark of UNIX Systems Laboratories.

---

Written by Mark Stevens, Ananda Bantari, and Lance Lickus.

# Contents

Introduction .....	1
Installation.....	2
SDS Products.....	4
Getting Started .....	5
The SingleStep Debug Window.....	8
The Panel Program .....	9
Seeing an Embedded Program Run.....	12
Looking at the Details.....	16
Variables and Commands .....	20
The Read Window.....	23
Index .....	25

