```
REVISION HISTORY : BETA debugger
--------------------------------

Note: Many of these revisions are for internal tech-support reference
and were not actually released.

4.XX            all have C and assembler expression evaluator,
                new variable and watch window (see below for details).
                These debuggers are 32 bit DOS executables using a
                DOS extender.

4.53 14/06/95   allows alternate register window layout
                (use /R switch on command line or press Alt-R in debugger)
                (for Sean @ Core Design)

4.54i 19/06/95  new resetcpu code - not much use until we've modded DEXBIOS
                   to force a re-load of SNPATCH though (soon).
      22/06/95  hex No.s in C exprs don't need leading zero anymore
                memory ranges of configs other than .C00 ignored
                hopefully fixed occasional missing cursor on alt screens
                allows modification of variables in Watch and Var windows
                hard break option now supports instruction fetch (PC) detection
4.55i 23/06/95  Oops. re-fixed expression eval to handle known symbols OK.
                fixed old error message problem for unrecognised symbol

4.60 05/07/95   dereferencing a pointer now fixes class to STATIC
                (so deref of a watchpoint register pointer to a structure is
                now correct)

4.61 06/07/05   changing screens no longer flushes message buffers
                message buffer increased to 100 lines.

4.62 06/07/95   auto-detects C startup and runs to main() if not given /d command line
 switch

4.63 26/07/95   minor update: fixed shift-f6 (clear all breakpoint counts)

4.64 28/07/95   restricted auto-source-mode, to only if no config file
                (because it was annoying developers)

4.65 28/07/95   restricted auto-source-mode, to only if no config file
                (but still got auto-run to main)

4.67 09/07/95   removed 'makesafe' call from CPE loader
                now *does not* turn off ints before program starts

4.69 24/07/95   now supports source file paths (Alt-P in file window)
                Alt-L and Ctrl-L no longer crash Var and Watch windows

4.70 01/09/95   Automatic Overlay Support (see also OVLEGPSX.ZIP for example)
                Unix \n newlines now work properly in message window

4.71 14/09/95   + fixed name completion that I broke in 4.70

4.73 21/09/95   fixed memfollow (alt-F in memory window to follow ptr)

4.74 21/09/95   mouse click in zoomed window now OK
                cursor clamping to left of mem window corrected.

4.75 26/09/95   stack-crawl (left and right arrow in var window)
                cexp.asm seperated from main code (as MASM couldn't cope)
                'searching' box in file window now hides if not found
                update animation only active if continous update mode
                condition on hardware break that was done for Williams in 4.65s is now
 back

4.76 28/09/95   Unix (LF) or DOS (CRLF) or MAC (CR) newlines now
                 work properly in message window. (from 4.70 the DOS
                 ones were accidentally double-spaced.)

4.76a29/09/95   allows $ in C names (for Carolyn @ EA)
                <- and -> starck crawl so array index changing is
                 now relocated to < and > (or , and .)
                top level of Var window now leaps to C callstack display

4.77 16/10/95   Pseudo-support for long long (64 bit) data type

4.78 06/11/95   can now display float and double data in watch & var windows

4.79 13/11/95   fixed DOS extender quirk which was causing
                'UPLOAD MEM TO FILE' to fail for amounts >4K (ver 4.78 only)

4.80 15/11/95   changed shift-esc makesafe so that ints *stay* off
                modded ctrl-F2 'prog reload' so breakpoints are preserved.

4.81 21/11/95   restored arrow keys in Var window to change array index
                C-callstack-crawler now on < and > (or comma and dot)

4.82 24/11/95   C expression evaluator now supports typecasting

4.83 14/12/95   OK, I've neatened up some key assignments used in
                stack browsing. See C callstack browsing below for
                details.

4.84 15/02/96   ctrl-F9 to restore IRQ status (to what it was b4 shift-esc) and run
                 + some minor dialog changes

4.85 15/02/96   space bar toggles window between source & dis modes

4.86 16/02/96   global scope can now be seen whilst stepping overlays

4.87 26/02/96   corrected enum display in watch/var window

4.88 07/05/96   new /k switch forces makesafe before CPE downloads

4.89 06/06/96   fixed minor diassembly syntax error.

4.90 25/07/96   fixed missing locals for huge C++ programs
                 with lots of overlays

----------------------------------------------------

Copy your old DBUGPSX.EXE away somewhere safe first cos you may need
to switch back to it if you have problems.

Please delete any old debugger config files (from debuggers prior to
version 4.0) before running the new debugger (or you will get
horrible screen colours and worse).

You should copy one of this EXE files to DBUGPSX.EXE in your path
(wherever you normally keep DBUGPSX.EXE).
```

```
-------

V4.29 up allow limited hardware data breakpoints (alt-B)
        for C & Assembler (see later in this file)

-------

V4.25 up allows you to use the sizeof() function in a C expression.
This function will return the size of the result of the expression
within the brackets.

-------

This debugger features the all new VARIABLE window and WATCH window.
In either window you can use the following keys:-

+       open up the info (e.g. structure or array etc) under cursor

-       close ditto

tab     change result display format of C expression under cursor

right arrow     increments array index under cursor
left arrow      decrements array index under cursor

Note: currently these last two only work if the array entry has not
been expanded by pressing '+'.

Also in a watch window you can use INSERT and DEL to add and remove
watch expressions. Pressing Alt-G will enter watch records for all
your global variables. Unlike VAR window, these are saved on exit.

Note that watch expressions, window lock expressions, and calculator
expressions can now be C or assembler syntax - just click on the
button on the dialog box before you enter the expression to select
which type.

If you have any questions then please call or email me.

--
Andy Beveridge       andy@snsystems.co.uk


-------------------------------------------------------------------

HARDWARE BREAK - breakpoints on data references
-----------------------------------------------

Alt-B brings up a dialog to set a hardware break. You must have a
compatible downloader (version 4.02) installed to do this otherwise
the debugger will give an error message.

The Hard Break dialog has two modes. It should default to the last
mode it was used in.

In ASSEMBLY LANGUAGE mode the dialog will prompt you to enter a base
address and a mask value. The mask should contain a 1 bit where you
wish to compare the address bit. e.g. to detect a reference to one
specific byte you would set a mask of -1 ($FFFFFFFF). To detect a
reference to a word mask = -2 ($FFFFFFFE to ignore bit 0), etc
```

Note that because this is a base + mask detection rather than a range you can only detect a specific range if that range begins on a suitable boundary. e.g. bytes anywhere, shorts must be on even addresses, longwords must be on 4 byte boundaries etc

Specifically the block to protect must start at an address which is a multiple of the smallest power of two which is greater than the length of the range. But that's not a very convenient way to think about it - I prefer just to visualise the address bus bits in my head.

In C LANGUAGE MODE we hide the above from the developer and try to present something a bit more intuitive. Since the C compiler will always align shorts & words we do not have to worry about the above mentioned alignment restrictions for these sizes. So in C mode we can just accept a variable by name. No other parameters are necessary because the debugger will then calculate the address of the variable and assume a mask of -1, -2 or -4 depending upon the size of the variable. Working with something other than simple variables in C mode is a bit fiddly because internally we have to work with the hardware restrictions mentioned above. Aligning large data items like this is trivial in assembly language but not in C. Any suggestions for an alternative easy to use but flexible C programmer interface to this hardware would be very welcome.

In either mode you can turn a hard break off by clearing the flags for read and write detection. The read/write detection flags are simple toggles - just click on then with the mouse. I ought to have a keyboard interface for those too. I'll fix that up soon and add a clear button too (it would be redundant but is probably more intuitive).

THE NEW MESSAGE WINDOW for PSX printf() output

Some developers complained that printf() output received by DBUGPSX was too slow and had too small a buffer. To fix this I have implemented a character based (rather than block) message stream which is much faster and makes better use of buffer space. The new system also allows you to override the default buffer size.

The debugger now auto-detects which message system is being used and takes input from either.

So, to use the new turbo message system all you need to do is install the DEXBIOS or PSYBIOS (version 1.21 or later) and then the appropriate new message TSR.

SOURCE FILE PATHS - new with debugger version 4.69 up.

By typing Alt-P in a file window you can enter a search path to be used to find source files for source-level debugging.
This path works just like a normal DOS path - entries are delimited by semicolons.

When the debugger looks for source file it will check...

1) the full path to the file as in the symbol file. This will usually be a fully canonicalised DOS filename as seen by the machine upon which the project was built.

2) IF 1) above fails to find the file only then will it isolate the filename from the end of the built path-name and then search for that file in each of the directories specified in the source-file search path.

e.g. if original source file at build time was MAIN.C and it was compiled in the C:\PROJX\PSX\SRC directory then the .SYM file will contain the entry

C:\PROJX\PSX\SRC\MAIN.C to describe the file. The debugger will first seach for that specific file. If it cannot find it and your search path is set to:-

```
        C:\;C:\TEMP;H:\COMMON\SRC
```

then the debugger will next look for:-

```
   C:\MAIN.C  then  C:\TEMP\MAIN.C  then  H:\COMMON\SRC\MAIN.C
```

in that order. Note that you can specify the current directory in the search path just by adding an empty entry.

```
e.g.    ;C:\;C:\TEMP;H:\COMMON\SRC
```

will cause it to look in the current directory first

```
or      C:\;C:\TEMP;;H:\COMMON\SRC
```

will cause it to look in the current directory after C:\TEMP but before looking in H:\COMMON\SRC

If all is not clear then you are welcome to call or email SN Systems to ask for help.

AUTO OVERLAY SUPPORT - New from 4.70 up
==

This feature was added at the request of developers making very heavy use of overlays to handle converting extremely large PC games over to Playstation's limited 2MB RAM. It is still in development so if you have any suggestions for improvements please let us know. In particular, if you debug using multiple symbol files then we may need to make a few small changes. I'm holding off to see if anyone needs this.

Not much to say about this here as it is all automatic. To use this feature all you need is an appropriate symbol file that contains the new format overlay records. To make such a file you will need to be using a recent version of the linker which also has this support.

For more details check-out the OVLEGPSX.ZIP (OVerLay Example) file on our BBS and FTP site. This example will show you how to set-up your .LNK linker control file and .MAK makefile to produce symbol files with overlay information.

STACK CRAWL - New from 4.75 up
==============================

This feature can be accessed by:-

pressing < or > keys (or comma or dot) whilst in a VAR window.

Variable scope levels below the current scope (i.e. calling functions) can be selected by pressing > repeatedly to unpick the stack one function at a time.

A level above the default current scope (press '<' from current var scope) displays all the function calls on the call-stack including the parameters they were passed. You can select a scope context from that list for variable display (move cursor and press return).

Although this is a Var window key it will in fact temporarily change the cpu context as seen by all windows on the screen.
The right arrow key will dereference the current displayed scope by one level. i.e. it will change context to that of the routine which called the one you're in. It does this by unpicking the stack frame of the current routine - temporarily restoring registers, and program context to that of the instruction following the call to the current level. This can be done again and again to walk back up through the C call-stack . At each context you can examine that function's local variables and parameters. The register display will also change to that context - you will see that the PC is after the call.

Note that this only works whilst the target is stopped. If you continue the target running (or step or trace) then the context is first restored to the point at which you stopped the program.

If you dereference back to a point which has no scope then the debugger will allow this but the var window will correctly display that there is no scope for this context. All debugger windows will be correct for the new context - if you look at a disassembly window and go-to the PC (tab) and you will see that the PC is at the instruction which called the C function you unpicked. If you try to derefence again from this point you will see an error message informing you that there is no C stack frame to unpick at this point.

You should also be aware that at the beginning of a function the entry code may not yet have built it's stack-frame and therefore you cannot unpick the callstack at this point. The debugger will detect such attempts and show an error message indicating that the callstack is incomplete. If you step into the function from this point then you will be able to unpick the callstack again.