I assume that you have installed Psy-Q and the C compiler correctly
as documented in the readme file on the PSX distribution disks.
The following assumes that you have the DEXBIOS.COM TSR program
correctly installed in your PC (e.g. in your autoexec)

Not that the fileserver functions will perform much better if the
DEXBIOS command line includes a correct IRQ number.

e.g. DEXBIOS /aXXX /i15

DBUGPSX command line:-

Syntax:-

DBUGPSX </switches> <filename> <filename>...

fields enclosed <thus> are optional and can be in any order
filenames specify symbol files, default extension (.SYM) will be
added.

valid switches are:-

&expr<,..>	comma seperated list of parameter expressions
c-	turn case sensitivity off (see also c+)
efile	(filename optional) load CPE file into target
h	halt target at debugger startup
i###	specify update interval (in 1/18ths sec)
l#	set label level (0 to 4)
m+	force debugger to use official mouse driver
r##	(No. optional) override video bios data screen rows
sfile	override default setup file
t#	set target ID No. (default is 7)
u-	turn continual update mode off (see also u+)
vexprtext	evaluate exprtext and put result to stdout
x-	force debugger to run in real (not 386 protected) mode

KEYS THAT FUNCTION IN ALL WINDOWS:-

shift arrow	change current window (move to window in that direction).
F1	change current window. As above.
F2	move a window edge (to re-size windows)

F3	create new window (splits current window in two)
F4	Delete a window edge (merge two adjoining windows)
F7	STEP. If current window is source or disassembly then it will track the PC location. Works by placing a temporary breakpoint at next instruction.
F8	STEP OVER. This will step over the current instruction. If it is a JAL or BAL instruction this will cause the whole subroutine to be executed.
F9	RUN target. Starts the target CPU executing at the current PC location.
F10	Select MENU mode (from keyboard as opposed to mouse)
ESC	Stop the target program.
SHIFT-ESC	Stop the target program & halt interrupts. (DANGER!)

shift-f1set current window to specified type.

shift-f5clear all breakpoints

shift-f6reset all breakpoint counts

shift-f8put breakpoint after current instruction and run.

shift-f9run to (prompted for) address.

shift-f10	force debugger to reload config file
alt-U	toggle debugger window continous update on/off. I normally have this on but it does slow the target cpu slightly.
alt-I	set time interval (in 18th's second) for ditto updates) (target scan interval if updates are off).
alt-L	lock current window start to specified expression. e.g. I like to lock a tall narrow HEX window to 'sp' so that I can always see the current stack.
ctrl-L	temporarily toggle window lock active/inactive.
ctrl-S	save all cpu registers to temporary buffer

ctrl-R	restore all cpu registers from temporary buffer
alt-H	Hex Calc. Evaluate assembler type expression.
ctrl-F2	Reload & reset executable file (if one was specified on the debugger command line). Not recommended on PSX because of the pad_start() pad_stop() problem.
alt-V	display debugger Version information.
alt-X	eXit the debugger (saving debugger context)
/	force update of current window even if continual update mode is off.
*	force full screen update & redraw.
ctrl-X	eXit debugger without saving debugger config.
ctrl-F	Fill memory range with specified value.
ctrl-Z	Temporarily ZOOM current window to full screen. ctrl-Z again to return to window'ed mode.
alt-P	Print current window to LPT1
alt-NUM	instantly switch debugger config (including screen layout) to one of ten screens. (all context preserved at each switch)

MEMORY WINDOW SPECIFICS:-

alt-F	follow longword pointer at cursor.
<return>	enter expression to change value of location under cursor
0-9 A-F a-f	directly modify memory at cursor location
arrow keys	move cursor
pg-up pd-down	window up or down by one window-full

alt-W	toggle window display to bytes/words/longwords
+	increment memory location under cursor
-	decrement memory location under cursor
alt-G	move window to display at specified address
alt-S	search memory range for specified bytes/words/longs or ASCII text
alt-N	continue previous search.

DISASSEMBLY WINDOW SPECIFICS:-

alt-S	Search memory for specified instruction text
alt-N	continue ditto.
ctrl-D	disassemble memory range to a text file
left, right	adjust window start address
up, down	move cursor
pg-up, pg-down	as you'd expect
alt-G	Goto specified address
tab	Goto PC
shift-tab	Set PC to current cursor location
F5	toggle breakpoint at cursor
F6	run to cursor (temp breakpoint at cursor addr)
ctrl-C	attach trigger count to breakpoint at cursor
alt-C	attach trigger condition (expression) to breakpoint
<return>	single line assembler (not in PSX version yet)

REGISTER WINDOW SPECIFICS:-

left right up down	move cursor

0-9 A-F a-f	modify register under cursor
<return>	enter new value (expression) for register under cursor

WATCH WINDOW SPECIFICS:-

up down	move cursor
insert	insert new watch at cursor posn
del	delete watch at cursor position
+	Open up structure display (if selected watch expression is a structure.) This does not currently handle pointers to structures, just structures. This will improve with the new C expression evaluator.
-	Close ditto

VAR WINDOW SPECIFICS:-

as WATCH window but you cannot INSert or DELete in this window.

FILE WINDOW SPECIFICS:-

pg-up pg-down up down	move cursor
<enter>	display new text file
alt-S	search for text
alt-N	continue ditto
alt-T	set tabs for current file in current window
tab	locate to PC address
shift-tab	set PC to address of source line under cursor.
f7	step current source line

f8	step over current source line
f5	toggle breakpoint at cursor
f6	run until cursor
alt-G	goto address (prompts for expression)
alt-L	goto Line No of current file.

Message windows:

These display buffered text information from a specific debugger back-channel stream. The target writes text to a back channel stream by preforming a fileserver write to a negative file handle. -1 for first window, -2 for next etc.

These streams are buffered by the PC even when the debugger is not running.

See fileserver PCwrite() in LIBSN.H for further details.

VAR WINDOWS:

Display the variables which correspond to the current scope.
This is usually the local variables of the current function.

An example of the debugger in action:-

1) Change to the directory containing the BALLS example program

2) CCPSX -v -g -Xo$80010000 main.c -omain.cpe,main.sym

3) DBUGPSX main /e

This runs the debugger and loads the symbol file MAIN.SYM and loads the executable image of the same name (MAIN.CPE).

By default the debugger will display two windows. Register window (top) and disassembly window (bottom).

Press F3. Press UP arrow. This will create a new window.

Press shift-F1. Select Hex (press H or click with mouse).
This will make the current window a Hex window

Press shift-UP-arrow to move to the other (disassembly) window.

Press TAB to display the current PC if it is not already there.

Press F7 to step one instruction. F8 will step over subroutine calls.

Press alt-G, enter 'main<return>' in response to prompt.

Press F6 to run until the cursor address (start of function main)

PC is now in a region which the debugger has Source Info for so now
you can go to source level mode:-

Press shift-F1 and select File window. Just hit return in response to
the prompt for a filename. Now press TAB in the empty file window.
This will locate the source file line corresponding to the PC.

Now you can F7 and F8 to step your source code.

Press F9 to just run the program. Notice that the source code
contains a pollhost() function call. (SEE LIBSN.H)
This allows the debugger to access the PSX memory and cpu even
though the user application is running. Your code should always have
a pollhost() somewhere active if you wish to access the target whilst
it is running.

You may also want to try creating more windows using F3 and re-sizing
them using F2. Delete window edges using F4 etc. Notice that these
are the same keys used by Brief to manipulate windows.

Notice that if you create a VAR window it will display the variables
curently in scope (or globals if there is no current scope). The
scope will change as you step into and out of different functions.

FILESERVER FUNCTIONS:-

The fileserver allows you to access the PC filing system from the PSX
to load data files to memory and also to write files on the PC filing
system and to access the Psy-Q debug back-channel to display debug
messages in debugger windows.

Don't forget to #include <libsn.h> if you wish to use these

functions.

Here are the prototypes for the functions in LIBSN:-

```
/*
** LIBSN.H    declare libary functions provided by LIBSN.LIB
**
** 05/02/94 ADB
** 21/03/94     ADB    added user notes as comments
** 18/09/94 ADB        added PCcreat() - it was missing before
*/

#definepollhost()       asm("break 1024")      /* inline to keep variable scope */


/*
** FILESERVER FUNCTIONS:
**
** NOTE: For PCread and PCwrite do not load files by passing extreme
** values for count as you might on UNIX as this will cause the full
** amount specified to be transferred - the file will be padded to
** that length with zeroes which may over-write memory beyond the
** end of the file.
**
** If you are unsure of the length of a file which you are about
** to read into memory then perform a
**        len = PClseek( fd, 0, 2);
** This will set len to the length of the file which you can then
** pass to a PCread() function call.
**
*/

/*
** re-initialise PC filing system, close open files etc
**
** passed: void
**
** return: error code (0 if no error)
*/
int        PCinit (void);

/*
** open a file on PC host
**
** passed:        PC file pathname, open mode, permission flags
**
** return:        file-handle or -1 if error
**
```

```
** note: perms should be zero (it is ignored)
**
** open mode:0 => read only
**                                  1 => write only
**                                          2 => read/write
*/
int       PCopen (char *name, int flags, int perms);

/*
** create (and open) a file on PC host
**
** passed:       PC file pathname, open mode, permission flags
**
** return:       file-handle or -1 if error
**
** note: perms should be zero (it is ignored)
*/
int       PCcreat (char *name, int perms);

/*
** seek file pointer to new position in file
**
** passed: file-handle, seek offset, seek mode
**
** return: absolute value of new file pointer position
**
** (mode 0 = rel to start, mode 1 = rel to current fp, mode 2 = rel to end)
*/
int       PClseek (int fd, int offset, int mode);

/*
** read bytes from file on PC
**
** passed: file-handle, buffer address, count
**
** return: count of number of bytes actually read
**
** note: unlike assembler function this provides for full 32 bit count
*/
int       PCread (int fd, char *buff, int len);

/*
** write bytes to file on PC
**
** passed: file-handle, buffer address, count
**
** return: count of number of bytes actually written
**
** note: unlike assembler function this provides for full 32 bit count
```

```
*/
int       PCwrite (int fd, char *buff, int len);

/*
** close an open file on PC
**
** passed: file-handle
**
** return: negative if error
**
*/
int       PCclose (int fd);
```