

# **Systems Manual**

## **PC Development System for the Nintendo 64**

**© SN Systems Software Ltd.**

## **END USER LICENSE AGREEMENT BETWEEN THE “LICENSEE” AND SN SYSTEMS LTD “LICENSOR”**

**LICENSE:** SN Systems Ltd (SN Systems) hereby grant the Licensee a non-transferable, non-exclusive right to use the Licensor’s software product Tools on any single computer, provided that the Software is in use on only one computer at a time in return for the license fee.

**USE OF THE SYSTEM:** You may use the Software and associated User Documentation on any single computer fitted with Cartridge Hardware. You may also copy the Software for archival purposes, provided that any copy contains all the proprietary notices for the original Software.

You may not:

Permit other individuals to use the Software except under the terms listed above;

Modify, translate, reverse engineer, decompile, disassemble (except to the extent applicable laws specifically prohibit such restriction) or create derivative works based on the Software;

Copy the Software (except for backup purposes);

Rent, lease, transfer or otherwise transfer rights to the Software;

Remove any proprietary notices or labels on the Software

**TITLE:** Title, ownership rights and intellectual property rights in and to the software shall remain in SN Systems Ltd.

**COPYRIGHT:** The Software is owned by the Licensor. The Licensee may not copy the manual (s) or any other written materials accompanying the Software.

**LIMITED WARRANTY:** The Licensor warrants that the Software will perform substantially in accordance with the accompanying manual (s) for a period of 30 days from the date of receipt PROVIDED that the failure of the Software has not resulted from accident, abuse or misapplication.

**CUSTOMER REMEDIES** The Licensor’s entire liability and the Licensee’s exclusive remedy shall at the Licensor’s option, either be:

(1) return of the license fee paid or

(2) repair or replacement of the Software that does not meet the Licensor’s Limited Warranty outlined above.

**DISCLAIMER OF WARRANTY: THE SOFTWARE, ACCOMPANYING MANUAL (S) AND ANY SUPPORT FROM SN SYSTEMS ARE PROVIDED “AS IS” AND WITHOUT ANY OTHER EXPRESSED OR IMPLIED WARRANTY, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL SN SYSTEMS BE LIABLE FOR ANY DAMAGES, INCLUDING LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF SN SYSTEMS IS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR FOR ANY CLAIM BY YOU OR ANY THIRD PARTY. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THE AGREEMENT.**

SN Systems’ liability under this Agreement whether in contract, tort (including negligence) or otherwise shall be limited to the Fee paid by the Licensee.

**TERMINATION:** This license will terminate automatically if you fail to comply with the limitations described above or if after thirty (30) days written notice to SN Systems, you terminate it. On termination you must destroy all copies of the Software, in whole or in part, in any form and cease all use of the Software.

**Please note that as this software is constantly being updated, it is quite likely that this manual may contain some inaccurate or out-of-date references and some features of newly updated software may not be fully covered.**

**For this reason, if you experience any difficulties with this documentation, updates are available for download via the SN Systems BBS.**

**We recommend that you make regular use of this service and quickly take advantage of any new features added to the software, report or download 'bug' reports, gain answers to questions that may be causing you difficulty and keep up-to-date on news concerning the development industry.**

**If you experience any difficulties, please do not hesitate to contact our Technical Support at SN Systems:**

**Tel: +44 (0)117 929 9733**

**Fax: +44 (0)117 929 9251**

**This device complies with part 15 of FCC Rules. Operation is subject to the following two conditions:**

- 1) This device may not cause harmful interference**
- 2) This device must accept any interference that may be received, including interference that may cause undesired operation.**

**© 1996, 1997 SN Systems Software Ltd. All rights reserved.**



<b>INTRODUCTION.....</b>	<b>I</b>
ABOUT THIS SYSTEM.....	II
THE SYSTEM FOR NINTENDO 64 .....	III
ISSUE INFORMATION.....	IV
ACKNOWLEDGEMENTS.....	V
<b>INSTALLATION.....</b>	<b>1-1</b>
INSTALLING THE HARDWARE.....	1-2
INSTALLING THE PC INTERFACE.....	1-4
INSTALLING THE PC SOFTWARE.....	1-9
INSTALLING THE GNU 'C' SOFTWARE.....	1-11
CHECKING THE INSTALLATION AND FIRST-TIME USE OF THE SYSTEM.....	1-12
REQUIREMENTS FOR DEBUGGING CARTRIDGE SOFTWARE.....	1-14
ADDITIONAL NOTES .....	1-15
ELFCONV - LIBRARY CONVERTER PROGRAM.....	1-16
<b>THE ASMN64 ASSEMBLER.....</b>	<b>2-1</b>
ASSEMBLER COMMAND LINE .....	2-2
ASSEMBLY ERRORS .....	2-5
<b>SYNTAX OF ASSEMBLER STATEMENTS.....</b>	<b>3-1</b>
FORMAT OF STATEMENTS .....	3-2
FORMAT OF NAMES AND LABELS.....	3-3
FORMAT OF CONSTANTS.....	3-4
SPECIAL CONSTANTS.....	3-6
ASSEMBLER FUNCTIONS.....	3-8
SPECIAL FUNCTIONS .....	3-9
ASSEMBLER OPERATORS.....	3-10
RADIX.....	3-12
ALIAS AND DISABLE.....	3-13
<b>GENERAL ASSEMBLER DIRECTIVES.....</b>	<b>4-1</b>
<b>ASSIGNMENT DIRECTIVES.....</b>	<b>4-2</b>
EQU.....	4-3
SET.....	4-4
EQUUS.....	4-6
EQR.....	4-8
RSIZE .....	4-9
RSSET.....	4-10
RSRESET.....	4-11
<b>DATA DEFINITION.....</b>	<b>4-12</b>
DSIZE.....	4-13
DCSIZE .....	4-14
DSSIZE .....	4-15
HEX.....	4-16
DATASIZE AND DATA .....	4-17
IEEE32 AND IEEE64.....	4-17
<b>CONTROLLING PROGRAM EXECUTION.....</b>	<b>4-18</b>
ORG.....	4-19
CNOP.....	4-20
OBJ AND OBJEND.....	4-21
<b>INCLUDE FILES.....</b>	<b>4-22</b>
INCLUDE.....	4-23
INCBIN.....	4-25

REF.....	4-26
DEF .....	4-27
<b>CONTROLLING ASSEMBLY.....</b>	<b>4-28</b>
END.....	4-29
IF, ELSE, ELSEIF, ENDIF, ENDC.....	4-29
CASE AND ENDCASE.....	4-31
REPT, ENDR.....	4-32
WHILE, ENDW.....	4-33
DO, UNTIL.....	4-34
<b>TARGET-RELATED DIRECTIVE.....</b>	<b>4-35</b>
REGS.....	4-36
<b>MACROS.....</b>	<b>5-1</b>
MACRO, ENDM, MEXIT.....	5-2
MACRO PARAMETERS.....	5-3
SPECIAL PARAMETERS.....	5-5
SHIFT, NARG.....	5-7
MACROS.....	5-8
PUSHP, POPP.....	5-9
PURGE.....	5-10
TYPE.....	5-11
<b>STRING MANIPULATION FUNCTIONS.....</b>	<b>6-1</b>
STRLEN.....	6-2
STRCMP.....	6-3
INSTR.....	6-4
SUBSTR.....	6-5
<b>LOCAL LABELS.....</b>	<b>7-1</b>
SYNTAX AND SCOPE .....	7-2
MODULE AND MODEND.....	7-3
LOCAL.....	7-4
<b>STRUCTURING THE PROGRAM.....</b>	<b>8-1</b>
GROUP.....	8-2
SECTION.....	8-4
PUSHS AND POPS.....	8-6
SECT AND OFFSET .....	8-7
<b>OPTIONS, LISTINGS AND ERRORS.....</b>	<b>9-1</b>
OPT .....	9-2
ASSEMBLER OPTIONS.....	9-3
OPTION DESCRIPTIONS.....	9-4
PUSHO AND POPO.....	9-7
LIST AND NOLIST.....	9-7
INFORM AND FAIL .....	9-9
XDEF, XREF AND PUBLIC.....	9-10
GLOBAL.....	9-11
<b>DEBUGGER FOR WINDOWS 95.....</b>	<b>10-1</b>
<i>Introduction.....</i>	<i>10-1</i>
<i>Projects.....</i>	<i>10-1</i>
<i>Views.....</i>	<i>10-1</i>
<i>Colour Schemes.....</i>	<i>10-2</i>
<i>Files.....</i>	<i>10-2</i>
<i>Dynamic Update.....</i>	<i>10-2</i>
ON-LINE HELP AVAILABLE FOR THE DEBUGGER.....	10-3

---

INSTALLING THE DEBUGGER.....	10-4
<i>Directory Structure.....</i>	<i>10-4</i>
OBTAINING RELEASES AND PATCHES .....	10-5
<i>Determining The Latest Releases And Patches.....</i>	<i>10-5</i>
<i>Mailing Lists.....</i>	<i>10-5</i>
<i>Addresses for SN Systems' ftp, web and BBS sites.....</i>	<i>10-5</i>
<i>Beta Test Scheme.....</i>	<i>10-6</i>
INSTALLING A FULL RELEASE.....	10-7
UPGRADING YOUR SYSTEM.....	10-8
CONFIGURING YOUR SCSI CARD .....	10-9
TESTING THE INSTALLATION.....	10-10
<i>Documentation.....</i>	<i>10-12</i>
LAUNCHING THE DEBUGGER.....	10-15
THE FILE SERVER .....	10-16
<i>File Server Menu Commands.....</i>	<i>10-17</i>
CONNECTING THE TARGET AND UNIT.....	10-18
PLUG-IN COMPONENTS.....	10-19
USING THE DISASSEMBLE MEMORY DIALOG.....	10-22
USING THE UPLOAD/DOWNLOAD MEMORY TOOL .....	10-24
PROJECTS.....	10-26
<i>Setting Up And Managing Projects.....</i>	<i>10-26</i>
SELECTING FILES FOR YOUR PROJECT .....	10-27
<i>Adding Files To The List Of Project Files.....</i>	<i>10-27</i>
CHANGING THE ORDER OF FILES IN THE FILE LIST .....	10-28
SPECIFYING CPE FILE PROPERTIES.....	10-29
SPECIFYING SYMBOL FILE PROPERTIES.....	10-30
SPECIFYING BINARY FILE PROPERTIES.....	10-31
SAVING YOUR PROJECT .....	10-32
RE-OPENING A PROJECT .....	10-32
SAVING A PROJECT UNDER A NEW NAME.....	10-33
RESTORING A PROJECT.....	10-33
OPENING AN EXISTING PROJECT .....	10-34
MANUALLY LOADING FILES INTO A PROJECT .....	10-35
THE DEBUGGER PRODUCTIVITY FEATURES.....	10-36
TOOLBAR ICONS.....	10-36
HOT KEYS.....	10-37
VIEWS.....	10-38
CREATING A VIEW .....	10-39
CYCLING BETWEEN VIEWS.....	10-40
SAVING YOUR VIEWS.....	10-41
NAMING A VIEW .....	10-41
CHANGING COLOUR SCHEMES IN VIEWS.....	10-42
WORKING WITH PANES.....	10-44
SPLITTING PANES .....	10-44
CHANGING PANE SIZES .....	10-45
DELETING A PANE.....	10-45
CHANGING FONTS IN PANES.....	10-46
SCROLLING WITHIN A PANE .....	10-47
SELECTING A PANE TYPE .....	10-48
MEMORY PANE.....	10-49
REGISTERS PANE .....	10-51
DISASSEMBLY PANE.....	10-52
SOURCE PANE.....	10-54
<i>Changing Source Files In The Source Pane.....</i>	<i>10-55</i>
<i>Navigating Source Files In The Source Pane.....</i>	<i>10-56</i>
LOCAL PANE .....	10-57
WATCH PANE.....	10-59
<i>C Type Expressions In Watch Pane.....</i>	<i>10-61</i>
<i>Assigning Variables.....</i>	<i>10-62</i>

---

<i>Expanding Or Collapsing A Variable</i> .....	10-63
<i>Traversing An Index</i> .....	10-64
<i>Adding A Watch</i> .....	10-65
<i>Editing A Watch</i> .....	10-66
<i>Deleting A Watch</i> .....	10-67
<i>Clearing All Watches</i> .....	10-67
DEBUGGING YOUR PROGRAM.....	10-68
SPECIFYING THE POLLING RATE AND CONTINUAL UPDATE RATE .....	10-69
<i>Forcing An Update</i> .....	10-70
USING THE EXPRESSIONMANAGER.....	10-71
SETTING BREAKPOINTS.....	10-77
USING THE CALL-STACK DISPLAY.....	10-83
STEPPING INTO A SUBROUTINE.....	10-86
STEPPING OVER A SUBROUTINE.....	10-87
STEPPING OUT OF A SUBROUTINE.....	10-88
RUNNING TO THE CURRENT CURSOR POSITION.....	10-89
RUNNING PROGRAMS.....	10-90
STOPPING A PROGRAM RUNNING.....	10-90
MOVING THE PROGRAM COUNTER.....	10-91
MOVING THE CARET TO THE PC.....	10-92
MOVING TO A KNOWN ADDRESS OR LABEL.....	10-93
EXPRESSION EVALUATION FEATURES .....	10-95
<i>Register Names</i> .....	10-95
<i>Typecasts and Typedefs</i> .....	10-95
<i>Labels</i> .....	10-96
<i>Functions</i> .....	10-96
<i>Expression Evaluation Name Resolution</i> .....	10-96
PREVIOUSLY ENTERED EXPRESSIONS HISTORY LIST .....	10-97
ANCHORING PANES TO THE PC.....	10-97
ANCHORING MEMORY PANES.....	10-98
IDENTIFYING CHANGED INFORMATION.....	10-99
CLOSING THE DEBUGGER WITHOUT SAVING YOUR CHANGES .....	10-99
CLOSING THE DEBUGGER AND SAVING YOUR CHANGES.....	10-100
<b>THE PSYLINK LINKER.....</b>	<b>11-1</b>
PSYLINK COMMANDLINE .....	11-2
LINKER COMMANDFILES .....	11-4
GLOBAL.....	11-6
XDEF, XREF AND PUBLIC.....	11-7
<b>THE PSYLIB LIBRARIAN.....</b>	<b>12-1</b>
PSYLIB COMMANDLINE .....	12-2
<b>THE CCN64 BUILD UTILITY.....</b>	<b>13-1</b>
CCN64 COMMANDLINE.....	13-2
SOURCE FILES .....	13-4
<b>THE PSYMAKE UTILITY.....</b>	<b>14-1</b>
PSYMAKE COMMANDLINE .....	14-2
CONTENTS OF THE MAKEFILE.....	14-3
TBIOS2.COM.....	A-1
RUN.EXE - PROGRAM DOWNLOADER.....	A-3
RUNNING WITH BRIEF.....	A-4
ASSEMBLER ERROR MESSAGES .....	B-1
PSYLINK ERROR MESSAGES.....	B-13
PSYLIB ERROR MESSAGES.....	B-18
<b>INDEX.....</b>	<b>I-ERROR! BOOKMARK NOT DEFINED.</b>







# Introduction

This system is a fast PC based cross development system for producing and testing mixed C and/or assembler programs for the Nintendo 64 games console.

## This version of the system features:

- High performance SCSI interface card for host PC
- Compact Nintendo 64 adapter cartridge this includes 32 Mbytes of cartridge emulation RAM
- All the software you need:-
  - A RISC R4300 assembler compatible with standard C compilers including the popular Freeware Gnu-C.
  - Fast R4300 assembler with numerous directives.
  - High Speed Linker and Librarian, with extensive link-time options.
  - A powerful Source Level Debugger which allows the programmer to step, trace and set breakpoints directly in the source code.

## Additional hardware required

- Host 386/486/Pentium PC with hard disk drive **at least 16 Megabytes of memory** and 1 free 16 bit ISA slot.
- Nintendo 64 and a valid games cartridge.

## Additional software required

- Windows 95 with Microsoft Internet Explorer.

## About This System

- This system has been developed by **SN Systems Ltd** who have many years experience of development software and developers' needs. It represents the next generation of development systems, backed up by a commitment to continual enhancement, development and technical support.
- The system includes two industry-standard Assemblers, a Linker, a Debugger and a C and C++ compiler. The Assemblers are extremely fast, and fully compatible with other popular development systems.
- It offers Source Level Debugging. This allows you to step, trace, set breakpoints, etc. in your original C or Assembler source code. The system automatically and invisibly, handles multiple text files.
- **It** provides a high-speed genuine SCSI parallel link between the Host PC and target system, with a data transfer rate of over 1 Megabytes per second. The system supports up to 7 connected target devices

## The System for Nintendo 64

The target interface is a compact cartridge that plugs into the cartridge port of the Nintendo 64.

Adapter firmware provides diagnostics and self-test facilities. Also included are assorted functions for useful run-time control of the development environment, as well as extensive file server facilities which allow the target to manipulate files on the host PC.

## Issue Information

**Compatible** development systems are available for a variety of other platforms:

- Sony PlayStation
- SEGA Saturn
- SEGA 32X
- SEGA MegaDrive/Genesis
- SEGA Mega-CD
- Nintendo Super NES
- Commodore Amiga 1200 and 600
- Williams Phoenix Arcade Board

### Nintendo Development Software

The software for Nintendo 64 comes on four 3.5" HD disks. Disk 1 and 2 contain all the System software, including the R4300 Assembler, Debugger etc. Disk 3 and 4 contain an archive of the 'Freeware' GNU C compiler.

**Note:** A number of Nintendo 64 specific Libraries are provided by Nintendo.

## Acknowledgements

### DOS and Windows

Microsoft, MS, MS-DOS are registered trademarks of Microsoft Corporation; Windows and Windows 95 are trademarks of Microsoft Corporation.

**IBM** IBM is a trademark of International Business Machines

**Brief** Brief is a trademark of Borland International.

**Sony** Sony PlayStation is a trademark of Sony.

**Nintendo** Super NES is a trademark of Nintendo Corporation.  
Nintendo 64 is a trademark of Nintendo Corporation.





## CHAPTER 1

# Installation

The development system consists of the following physical components:

- **PC Board**
- **Target Interface**
- **SCSI Connecting Cable**
- **Mains Connecting Cable**
- **Software**
- **Universal Power Supply - IEC Socket, rating 100-250v a.c.,47-63 Hz**

**Warning:** For the Nintendo 64 Development Cartridge you **must** use the supplied 5V regulated power pack. Use of any other type will cause serious damage to the Unit.

Installation is, therefore, a relatively straightforward procedure, and is described in this chapter under the following headings:

- **Installation The Hardware**
- **Installing The PC Interface**
- **Installing The PC Software**
- **Installing The GNU C Software**
- **Checking The Installation And First Time Use Of The System**
- **Requirements For Debugging Cartridge Software**
- **Additional Notes**
- **Elfconv - Library Converter Program**

## Installing The Hardware

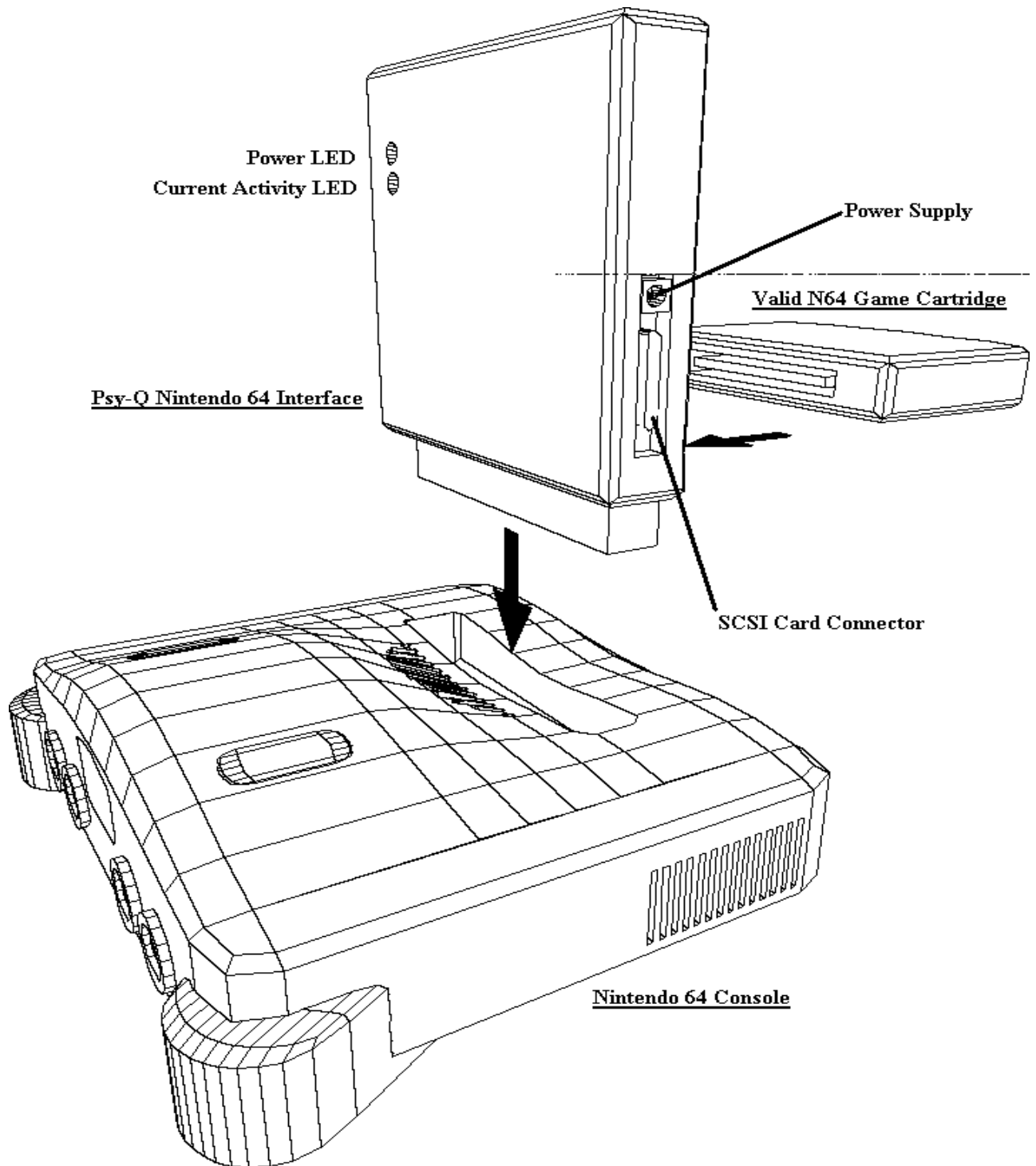
1. Check the configuration of the SCSI Card and install in the host PC; see later in this chapter for full installation details.

**Note:** Ensure that you record the **IO Address** and **IRQ** value as these will be required when you install the software.

2. Plug a standard Nintendo 64 game cartridge (e.g. Mario64 or Pilotwings64) into the through connector at the back of the unit, so that the cartridge is front face up when the interface is inserted in the Nintendo 64. The cartridge is required to provide the security chip and the 4k boot header block. You should also be able to make use of any additional hardware on the cartridge, e.g. game save memory.
3. Plug the interface into the Nintendo 64; the LED should be towards the front.

**Note:** Ensure that the Nintendo 64 is switched **off** at this point.

4. Connect the SCSI cable to the PC.



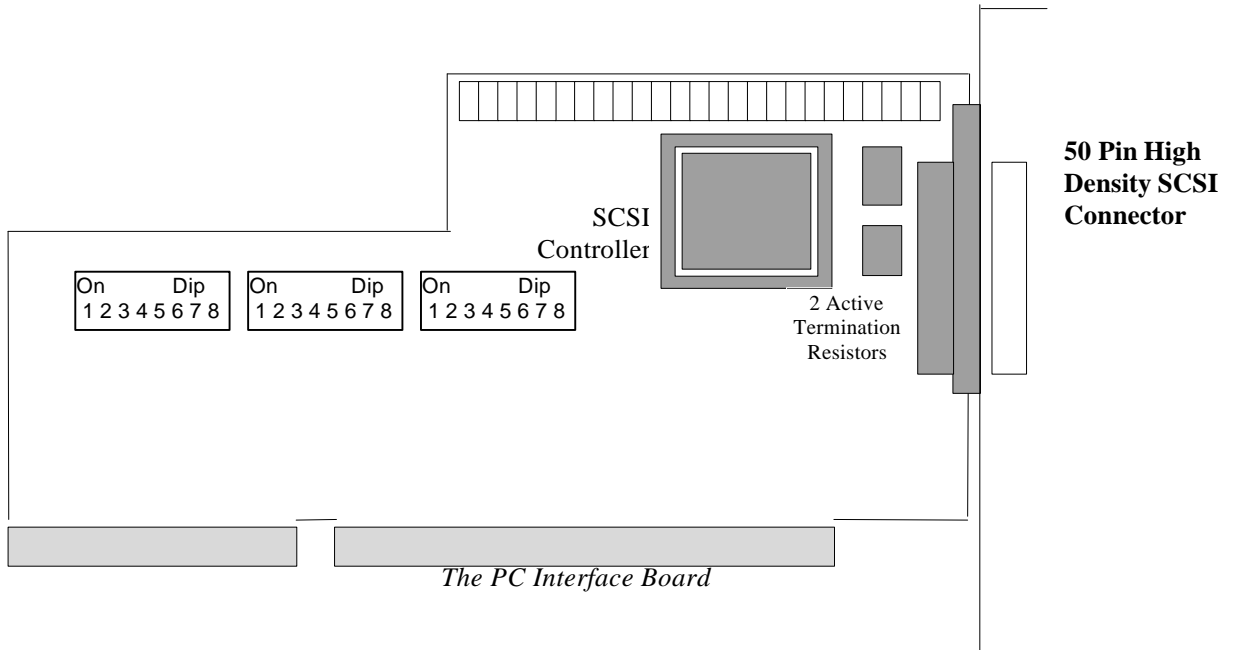
5. Plug the power supply into the side of the interface. The power LED (the top one) should come on and the lower LED should flash. Check the hardware if the lower LED is not flashing.

**Note:** At this point the Nintendo 64 should **still** be switched off.

## Installing the PC Interface

The PC Interface board should be fitted in to an empty 16 bit slot in the host PC. The host must be an IBM PC-AT or compatible, running under MSDOS 3.0 or better.

Prior to fitting, the 3 banks of dip switches should be checked and configured as required. It is likely, however, that the factory setting will suffice. They are described below.



**CAUTION:** This board is sensitive to static electricity; hold by the metal support bracket when handling.

## DIP SWITCHES

The PC card is configured by altering the three dip switches on the card. These switches alter:

**DMA**

**IRQ**

**SCSI Termination Power**

**IO Address**

**SCSI ID** for the card (normally on ID 7)

FUNCTION	DEFAULT
DMA 7	On, On
DMA 6	Off, Off
DMA 5	Off, Off
IRQ 15	On
IRQ 12	Off
IRQ 11	Off
IRQ 10	Off
IRQ 7	Off
IRQ 5	Off
Not Used	Off, Off
SCSI Termination Power	On
IO Address - A3 - A8	Off, On, On, On, On, Off
Card SCSI ID	On, On, On

## DMA

DMA or Direct Memory Access, is a mechanism for the fast and efficient transfer of data (in either direction) between the SCSI card and the PC's memory.

This transference occurs via DMA channels **5, 6 or 7**.

Only one channel can be used per card; if more than one card is in use a different DMA channel must be used for each one.

**Note:** Data transfer may be interrupted if the same channel is used for more than one card.

A channel is selected by switching the pair of adjacent dip switches **On** and the remaining pairs to **Off**.

The default setting is 7; this is achieved as follows:

<b>DMA 7 - On, On</b>
<b>DMA 6 - Off, Off</b>
<b>DMA 5 - Off, Off</b>

**Note:** In each pair, both switches must be set to the same value.

**Note:** If all DMA dip switches are set to Off, the transfer of data will be slower because this process will instead be carried out by the PC Processor

## IRQ

An Interrupt Request (IRQ) number can be assigned to each attached device so that it can quickly interrupt the PC Processor with a request.

When the PC receives an Interrupt Request it goes to a look-up table and ascertains the purpose of the sub-routine attached to the IRQ number; it then processes this sub-routine as soon as possible.

The SCSI Card offers the following IRQ numbers.

**15, 12, 11, 10, 7, 5**

Select the required number from this list by switching the adjacent dip switch **On**.

**Note:** You can **not** use an IRQ number which is being used by another device. The following procedures will list those currently in use:

1. From the Start menu select the **S**ettings option.
2. Select **C**ontrol Panel.
3. Double-click the Systems icon.
4. Select the Device Manager tab.
5. Select the **P**roperties button.
6. Select the Interruptrequest (IRQ) button.  
The IRQ numbers currently in use will be displayed.

**Note:** The default IRQ setting for the SCSI Card is **15**.

**Note:** It is possible to disable the IRQ number by removing all the jumpers but this may result in an impaired performance.

**Note:** The SCSI card setting must match that which will be later specified for the software.

**Note:** If you use a PCI Mother Board it may be necessary to make the IRQ available to the ISA Device in the BIOS. See the Mother Board documentation for further information.

### SCSI Termination Power

The SCSI Termination Power switch determines whether SCSI termination is supplied on the card. Correctly applied termination minimises electrical interference without it the communication between the devices would become erratic.

The SCSI devices are connected in a chain structure. Termination is only required by the devices at each end of the chain.

The default setting for this configuration is **On**.

In normal operation this should **not** be changed.

**IO Address** The IO Address provides a channel of communication between the SCSI card and the PC.

You must first set the card to a hardware IO Address and then set the software to access it at that address.

**Note:** If the hardware address does not match the software setting, a message will indicate that the card is not found at that address.

The card offers a very large range of IO addresses from **200<sub>16</sub> to 3f8<sub>16</sub>** in increments of 8. The address is changed by altering the 6 dip switches labeled A3 to A8.

**A8** is the most significant bit, and **A3** is the least.

An address line is selected by switching it to **Off**.

The default setting is **308**; this is set as follows:

**IO Address A3 - A8 - Off, On, On, On, On, Off**

Some examples are shown in the following table:

	A8	A7	A6	A5	A4	A3
240	On	On	Off	On	On	On
2A0	On	Off	On	Off	On	On
300	Off	On	On	On	On	On
308	Off	On	On	On	On	Off
310	Off	On	On	On	Off	On
318	Off	On	On	On	Off	Off
380	Off	Off	On	On	On	On
388	Off	Off	On	On	On	Off
390	Off	Off	On	On	Off	On
3E0	Off	Off	Off	Off	On	On

**Note:** You must not choose an address which **is the same as or within** the range of addresses occupied by another card as this would cause unpredictable results. Use the following procedures to list those currently in use:

1. From the Start menu select the Settings option.
2. Select Control Panel.
3. Double-click the Systems icon.
4. Select the Device Manager tab.
5. Select the Properties button.
6. Select the Inputoutput (I/O) button.  
The currently used addresses will be displayed.

**SCSI ID** Each attached device has its own **SCSI ID** which uniquely identifies it on the SCSI bus.

The last three switches are used to alter the **SCSI ID**.

The default setting is **7**; this is set as follows:

<b>Card SCSI ID - On, On, On</b>
----------------------------------

**Note:** You are advised not to change this setting (or re-use it for another card) as it has been pre-allocated in order to minimise any conflict with other internal boards. Any SCSI ID duplication would result in unpredictable results.



## Installing the PC Software

The installation discs contain programs to perform the following functions:

- **Assembling**
- **Linking**
- **Debugging**
- **Other Windows and miscellaneous tasks**



### To install the Development Tools:


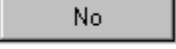
**Note:** The tools must be installed **before** the GNU C Compiler.

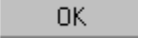
1. Insert the first development tools disc in the 'A' drive.
  2. Double-click the 'A' drive icon on your desktop and select the .exe file.
  3. Select  to view the installation help file or  to continue with the installation.
  4. Select  to confirm the Licence Agreement or  to abort the installation.
  5. The Select Components To Install dialog will be displayed. A tick will be shown alongside the Tools and the Debugger. Select  to install both items.
  6. From the Select Destination Directory Dialog, select  to confirm that you wish to place the Tools in the displayed directory or browse and select an alternative from the directory listing if required.
  7. To automatically set-up PSYQ\_PATH in autoexec.bat, select  at the Edit Autoexec dialog.
  8. From the Select Destination Directory Dialog, select  to confirm that you wish to place the Debugger in the displayed directory or browse and select an alternative from the directory listing if required.
- Note:** At this point you will **also** be required to specify a directory (or accept the default) for the GNU C Compiler.
9. Insert the second tools disc and select .
  10. Select  to insert a short-cut for the Debugger on the Start menu.

11. Set the required values for the SCSI card as follows:

Enter a 3 or 4-digit hexadecimal number to the **Port Address** and specify an **IRQ** value by clicking on the down arrow and selecting as appropriate.

**Note:** The values specified here must match those already specified on the hardware.

12. Select  or  at the Verify Values dialog.

13. Select  to restart your PC to complete the installation.

The Tools and Debugger are now installed.

## Installing the GNU 'C' Software

After you have installed the tools in their own directory on your hard disk you are ready to install the popular GNU 'C' N64 Compiler

The disk contains all of the files provided by GNU for the N64 'C' Compiler. For Nintendo 64 software development you may require components that can only be obtained directly from Nintendo. These include a number of useful Libraries and their C Header files which are specific to the Nintendo 64 environment. If you do not already have this software, you should contact Nintendo directly.



### To install the GNU C Compiler:

1. Insert the first GNU C Compiler disc to the 'A' drive.
2. Double-click the 'A' drive icon on your desktop.
3. Select the .exe file.
4. Select  if you agree to the Licence Agreement or  to exit the installation.
5. The Select Directory For GNU C Compiler dialog will be displayed. Select  to confirm the displayed directory or browse and select an alternative directory in which to place the GNU C Compiler.
6. Insert the second disc and select .

The installation is now complete.

**Note:** The library from Nintendo will have to be converted from **elf** format to the System's format. See the **elfconv** documentation below for further details about this process.

## Checking The Installation and First-Time Use Of The System



### Checking the Installation:

**Note:** Before proceeding with your installation check read the README file for any last minute changes or final release notes. This will be called README.HTM or README.RTF and will be found in the **psyq-win** directory.

1. At this point the Nintendo 64 should be switched off and the Target Adapter and game cartridge should be connected to it. The power should be on for the Adapter and the bottom LED should be flashing **slowly**, indicating that the bios has not yet been loaded.

**Note:** If the LED is flashing quickly this indicates that the Nintendo 64 is switched on and no bios is loaded. This is an error which **must** be rectified by switching the Nintendo 64 **off**.

2. Run the File Server by selecting the Debugger File Server from the Start menu.
3. This will attempt to connect to the Nintendo 64 console. If the connection is successful the File Server will download the bios; the bottom LED will go out and the following text will be output from the File Server:

**Psy-Q File and Message Server, Copyright 1996, 1997, SN Systems Ltd  
Version 2.0 (January 1997)  
Release 10, Patch Level 3**

**Target Found: Bus ID = 0, SCSI ID = 0  
New Downloader - Reading profile information...  
Profile read for Nintendo 64 (no bios)  
OK  
Rebooting the Nintendo 64 (no bios)...  
Getting the target's ID...  
ID is N64-NO-BIOS PLEASE LOAD  
bios1...  
sleep .5 sec...  
bios2...  
sleep .5 sec...  
New Downloader - Reading profile information...  
Profile read for Nintendo 64**

If this has occurred communication will have been established between the PC and the Nintendo 64.

4. Now you are ready to download a test cartridge image into the cartridge RAM.

Proceed as follows:

5. From the File Server select the **T**ools menu.
6. Select the **U**pload option.
7. Click the **D**ownload radio button and change the **D**ownload Address to 0xB0000000 .
8. Click **B**rowse and select the file `test.bin` from the `psyq-win\examples\n64` directory.
9. Click OK.

The downloaded image is now in the cartridge RAM.

10. Switch on the Nintendo 64. The system will boot the image and stop at a breakpoint at the start of the program.
11. Run the Debugger from the Start menu.
12. You are now ready to begin debugging.

**Note:** To convert the Nintendo 64 object files use the **Elfconv** converter program; this is described below.

## Requirements for Debugging Cartridge Software

In order to be able to debug your own program you must link in the debug stub code. This consists of two object modules occupying approximately 8k of ram. These are **PSYQ.OBJ** and **PSYQDBG.OBJ**. They contain only one entry point `'init_debug'` which must be called at the start of the program. The entry point will hook the debug vector at \$80000180. The programs will pass any interrupts/exceptions that they do not know how to handle to the previous address in this vector; therefore `'init_debug'` should not be called until this vector has been initialised, i.e. contains a valid address. The debug stub will not function until `'init_debug'` has been called; therefore any exceptions occurring before this will not be handled.

Further details can be found in the test program's Source and Linker Control Files

If the MIPS chip in the Nintendo 64 stops communicating with the microcontroller in the cartridge for any reason (e.g. the power is off or the program crashes badly), then the microcontroller will inform the Debugger (or any other program running on the PC and trying to communicate with the target) that the Nintendo is not responding and the error message **'Target Unit Is Busy'** will be displayed. If the target recovers or you reset it, the message will go away and the Debugger will continue to run.

The interface uses a hardware interrupt to force the MIPS chip in the Nintendo 64 to enter the debug stub. This means that it will not normally be necessary to add 'poll calls' to your program unless you need to break in at points where interrupts are disabled.

## Additional Notes

- The system has 32 Mbytes of ram fitted for cartridge emulation but the top 128k (0xB1FE0000 - 0xB1FFFFFF) is reserved for debug purposes.

**Note:** Do **not** load anything into this area or it will be erased.

- After you produce the cartridge image and before you load the games program into the Nintendo system, the program `setcsum.exe` will set the load/entry point of your program and the required checksum values. The minimum size for a cartridge is 1Mbyte+4Kbytes; if your code is smaller than this it will be padded to this length. The first 4k of the game image is a boot block. Most of the data in this block is shadowed through at Nintendo 64 boot up from the data in the cartridge you have plugged into the interface. `Setcsum.exe` will set any other required values so all you need do is have a 4k block of 0s at the start of your cartridge image.

**Note:** See the test program in `psyq-win\examples\n64` for an example of this.

- The debug stub disables all interrupts while it is executing.
- Although the Assemblers and Linker support the use of the instructions for manipulating 64 bit data (e.g. `ld`, `sd` etc), they still only contain a 32 bit expression evaluator so 64 bit constants cannot be used.
- The debug stub does not support the use of memory mapping the region `0x00000000 - 0x7FFFFFFF` or `0xC0000000 - 0xFFFFFFFF` via the TLB. Only 00s will be seen in these areas at the moment.

## ELFCONV - Library Converter Program

The Nintendo 64 libraries supplied by Nintendo contain object files in a file format known as ELF. These files must be converted to SN System's object file format if the libraries are to be used with SN development software. **Elfconv** is a utility program which performs this conversion.



### To convert a library

1. Three programs are required; one for each of the three stages of conversion:
  - A library tool which understands ELF libraries this will extract the object files from an ELF format library, creating one or more files with a `.o` extension.
  - `Elfconv`, this can convert all `.o` files found in the current directory, a single file or all files matching a given pattern. The output from `elfconv` is one or more files with a `.obj` extension.
  - SN System's PSYLIB librarian program this will combine one or more `.obj` files into an SN format library, suitable for linking with PSYLINK
2. The Nintendo 64 libraries are supplied for the SGI workstation the first program should therefore be the SGI's standard 'ar' library tool. On the SGI workstation, locate the library file(s) you wish to convert and type the following to extract all the object files from each library:

```
ar x libraryname
```

This will create one or more `.o` files in the current directory, reflecting the contents of that library.

3. Copy these files to your PC.
4. As `Elfconv`'s operation on each file is automatic, you only need to tell it which files to convert via one of the following methods:.

```
elfconv input output
elfconv input
elfconv /a
```

For the first method `elfconv` will search for the file with name `input` and convert it to a file named `output`.



If you specify only an input file, elfconv will convert the relevant file and create an output file with the same name but with `.obj` suffixed to its extension.

For the third method all files with `.o` extensions will be converted as per the second method.

Thus:

```
elfconv file1.o file1.obj    Converts file1.o into file1.obj
elfconv file1.o             Converts file1.o into file1.obj
elfconv /a                  Converts all .o files in the current
                             directory into .obj files
```

Elfconv will report the following kinds of errors

A missing input file  
An input file which is not recognisable as an ELF object file.  
Inability to create the output file (usually if there is no disk space).  
Fatal and non-fatal errors during conversion of the file; these start with "!!" for fatal and ">>" for non-fatal errors.

**Note:** You should not receive any fatal or non-fatal errors if elfconv is working correctly and you are converting a genuine N64 library file. If you do receive any errors, contact SN Systems for support.

5. After you have converted your object files, use PSYLIB to create an SN format library. See the PSYLIB chapter for further details.



## CHAPTER 2

# The ASMN64 Assembler

The **ASMN64 Assembler** can assemble source code at over 1 million lines per minute. Executable image or binary object code can be downloaded by the Assembler itself, to run in the target machine immediately, or later, by the **RUN** utility.

This chapter discusses how to run an assembly session, under the following headings:

- **Assembler Command Line**
- **Assembler and Target Errors**

## Assembler Command Line

During the normal development cycle, **ASMN64** may be:

- run in stand alone mode
- launched from an editing environment such as Brief - see later in the chapter
- invoked as part of the **Make** utility - see The Psymake Utility chapter.

When the Nintendo 64 Assembler is run independently, the command line takes the following form:

**Syntax**     **ASMN64** /switchlist source,object,symbols,listings,tempdata

or

**ASMN64** @commandfile

If the first character on the command line is a @ sign, the string following it signifies a command file containing a list of Assembler commands.

**Switches**     The assembly is controlled by inclusion of a set of switches, each preceded by a “forward slash (/)”. The “/o” switch introduces a string of assembler options; these can also be defined in the source code, using a **OPT** directive. Assembler options are described in detail in chapter 9, the available switches are listed below:

<b>/c</b>		Produce list of code in unsuccessful conditions
<b>/d</b>		Set Debug mode - if the object code is sent to the target machine, do not start it.
<b>/e</b>	<i>n=x</i>	Assigns the value <i>x</i> to the symbol <i>n</i> .
<b>/g</b>		Non-global symbols will be output directly to the linker object file.

<b>/j</b>	<i>pathname</i>	Nominate a search path for <b>INCLUDE</b> files.
<b>/k</b>		Permits the inclusion of pre-defined foreign conditionals, such as <b>IFND</b> - see also <b>MACROS</b> , chapter 5.
<b>/l</b>		Output a file for the Psylink Linker.
<b>/m</b>		Expand all macros encountered.
<b>/o</b>	<i>options</i>	Specify Assembler options - see chapter 9 for a full description of the available options.
<b>/p</b>		Output pure binary object code, instead of an executable image in <b>cpe</b> format - see also <b>RUN.EXE</b> chapter 2.
<b>/ps</b>		Output ASCII representation of binary file in Motorola <i>s-record</i> format
<b>/w</b>		Output <b>EQUATE</b> statements to the Psylink file.
<b>/z</b>		Output line numbers to the Psylink file.
<b>/zd</b>		Generate source level Debug information.

**Source** The file containing the source code; if an extension is not specified, the default is **n64**. If this parameter is omitted, the Assembler outputs *help* in the form of a list of switches.

**Object** The destination file to which object code is written.

**Symbols** The file to which symbol information is written, for use by the Debugger.

**Listings** The file to contain listings generated by assembly.

**Tempdata** This parameter nominates a file to be placed on the **RAM** disk for faster access. If the name is omitted, the default is **asm.tmp**; note that the temporary file is always deleted after assembly is complete.

**Remarks**

- If any of the above parameters are omitted, the dividing comma must still be included on the command line, unless it follows the last parameter.
- The Assembler run may be prematurely terminated by any of the following methods:
  - Pressing **Control-C**.
  - Pressing **Control-Break** (recognised more quickly because it does not require a **DOS** operation to spot it).
  - Pressing **Esc**.
- The Assembler checks for an environment variable called **ASMN64**. This can contain default options, switches and file specifications, (in the form of a command line), including terminating commas for unspecified parameters. Defaults can be overridden in the runtime command line.

**Examples**

```
asmn64 / zd / o ae+,w- scode,t0: ,scode.sym
```

This command will initiate the assembly of the R3400 source code contained in a file called **scode.n64**, with the following active options:

- *source level debug information* to be generated
- *automatic alignment* enabled
- *warning messages* to be suppressed
- the resultant *object code* to be transferred directly to the *target machine*, **SCSI device 0**
- *symbol information* to be output to a file called **scode.sym**
- *assembly listing* to be suppressed

```
ASMN64 @game.pcf
```

will recognise the preceding @ sign and take its command line from a command file called **GAME.PCF**.

## Assembly Errors

During the assembly process, errors may be generated as follows:

By the assembler itself, as it encounters error conditions in the source code.

### Remarks

**Appendix A** gives a full list of Assembler error messages.

plus

Abort, **R**etry or **B**us Reset





**CHAPTER 3****Syntax of Assembler Statements**

In order to control the running of an Assembler, source code traditionally contains a number of additional statements and functions. These allow the programmer to direct the flow and operation of the Assembler as each section of code is analysed and translated into a machine-readable format. Normally, the format of Assembler statements will mirror the format of the host language, and ASM64 follows this convention.

This chapter discusses the presentation and syntax of Assembler statements, as follows:

- **Format of Statements**
- **Format of Names and Labels**
- **Constants**
- **Functions**
- **Operators**
- **RADIX**
- **ALIAS and DISABLE**

## Format of Statements

Assembler statements are formatted as follows:

*Name or Label*      *Directive*      *Operand*

The following syntactical rules apply:

- Individual fields are delimited by spaces or tabs.
- Overlong lines can be split by adding an ampersand (&) and the next line is then taken as a continuation
- Lines with an equals (=) sign as the first character are considered to be the case options of a **CASE** statement - see Flow Control, chapter 4.
- Comment Lines
  - comments normally follow the operand, and start with a semicolon (;)
  - lines which consist of space or tab characters are treated as comments.
  - a complete line containing characters other than space or tabs is treated as a comment, if it starts with a semicolon or asterisk.

## Format of Names and Labels

Names and Labels consist of standard alpha-numeric symbols, including upper-case letters, lower-case letters and numeric digits:

**A to Z, a to z, 0 to 9**

In addition, the following characters can occur:

**Colon (:)** Can be used at the end of a name or label when defined, but not when referenced.

**Question Mark (?), Underscore (\_), Dot (.)**  
These three characters are often used to improve the overall readability

**AT sign @** Indicates the start of a local label - see chapter 7. Note that, by using the Assembler option `nl`, the local label symbol can be changed to a character other than @.

The following usage rules apply throughout:

- Numeric digits and Question Marks must not be the first character of a name.
- Labels normally start in column 1. However, if they start elsewhere, there must be no characters preceding the name, except space or tab, and the last character must be a colon.
- If a problem in interpretation is caused by the inclusion of a non-alphanumeric character in a Name or Label, that character can be replaced by a backslash, or the entire Name or Label surrounded by brackets.

## Format of Constants

The Assembler supports the following constant types:

### Character Constants

A character string enclosed in quote marks is a character constant and is evaluated as its ASCII value. Character constants may contain up to 4 characters, to give a 32 bit value. Thus:

"A"	= 65		= 65
"AB"	= (65*256)	+ 66	= 16706
"ABC"	= (65*65536)	+ (66*256) + 67	= 4276803
"ABCD"	= (65*16777216)	+ (66*65536) + (67*256) + 68	= 1094861636

### Integer Constants

Integer constants are normally evaluated as decimal, the default base, unless one of the following is used:

- The **RADIX** directive changes the base - see chapter 3.
- If %, \$ or 0x precede an **individual** integer, it signifies that the integer value will be evaluated as binary or hex:

% - Indicates that the base is binary, e.g. **%1001**

\$ - Indicates that the base is hex, e.g. **\$45af3921**

0x - Indicates that the base is hex, e.g. **0x45af3921**

Two possible characters are provided for base hex because option **r+** (either / or + on the command line or **opt r+** in the source), will allow the **\$** prefixed versions of **register** names to be used but will disable the use of **\$** to prefix **hex** constants. The **0x** prefix however, can still be used with this option, e.g:

```
opt r-
lw a0, $14 (t0)    ;or lw a0, 0x14 (t0)
                  ;or lw a0, 20 (t0)
```

This operation takes the address in register **t0**, adds the 14(hexadecimal) value to it and then takes the value at that address and puts it into **a0**.

This code is equivalent to:

```
opt r+
lw $4, 0x14 ($8)  ;or lw $4, 20 ($8)
```

- If a character is preceded by a backslash and up arrow (^) the corresponding control character is substituted.
- The **AN** Assembler option allows numbers to be defined as Intel and Zilog integers. That is, the number must start with a numeric character and end with one of:

**D** for Decimal; **H** for Hexadecimal; **B** for Binary

## Special Constants

The following pre-defined constants are available in ASMN64.

<b>_year</b>	As a two digit number, e.g. 95
<b>_month</b>	1 = January; 12 = December
<b>_day</b>	1 = 1st day of month
<b>_weekday</b>	0 = Sunday; 6 = Saturday
<b>_hours</b>	00 - 23
<b>_minutes</b>	00 - 59
<b>_seconds</b>	00 - 59
<b>*</b>	Contains the current value of the Location Counter.
<b>@</b>	Contains the actual PC value at which the current value will be stored - see below.
<b>narg</b>	Contains the number of parameters in the current macro argument - see chapter 5 for further details.
<b>__rs</b>	Contains the current value of RS Counter - see chapter 4 for further details.
<b>_filename</b>	A pre-defined string containing the name of the primary file undergoing assembly.

### Remarks **Time and Date Constants:**

Time and Date constants are set to the start of assembly; they are not updated during the assembly process.

**Example** `RunTime db\#_hours:\#_minutes:&  
                  \#_seconds"`

this expands to the form `hh:mm:ss`, as follows

```
RunTime    db    "21:08:49"
```

**Note:** This example uses the special macro parameter, `\#`, which is described in Chapter 5.

**Location Counter constants:**

The current value of the program pointer can be used as a constant. To substitute the value of the location counter at the current position, an asterisk (\*) is used:

```
Firstbss    section    Bss,g_bss
            equ        *
```

Since \* gives the address of the start of the line,

```
            org        $100
            dw         *,*,*
```

defines \$100 three times.

An @, when used on its own as a constant, substitutes the value of the location counter, pointing to an address at which the current value will be stored.

```
            org        $100
            dw         @,@,@
```

defines \$100,\$104,\$108.

## Assembler Functions

**ASMN64** offers many functions to ease the programmer's task. These are listed below; a more detailed explanation of their usage can be found later in this chapter. In addition, there are other specialised functions which are described in the following pages.

<i>Name</i>	<i>Action</i>
def( <b>a</b> )	Returns <i>true</i> if <b>a</b> has been defined
ref( <b>a</b> )	Returns <i>true</i> if <b>a</b> has been referenced
type( <b>a</b> )	Returns the data type of <b>a</b>
sqrt( <b>a</b> )	Returns the square root of <b>a</b>
strlen( <i>text</i> )	Returns the length of string in characters
strcmp( <i>text</i> <b>a</b> , <i>text</i> <b>b</b> )	Returns <i>true</i> if strings match
instr( <i>[start,]</i> <b>txa</b> , <i>txb</i> )	Locate substring <b>a</b> in string <b>b</b>
sect( <b>a</b> )	Returns the base address of section <b>a</b>
offset( <b>a</b> )	Returns the offset into section <b>a</b>
sectoff( <b>a</b> )	Equivalent to offset
group( <b>a</b> )	Returns the base address of group <b>a</b>
groupoff( <b>a</b> )	Returns the offset into group <b>a</b>



## Special Functions

**filesize("filename")**

Returns the length of a specified file, or -1 if it does not exist.

**groupsize(X)** Returns the current (not final) size of groupX.

**grouporg(X)** returns the **ORG** address of groupX or the group in whichX is defined ifX is a symbol or section name.

**groupend(X)** Returns the end address of groupX.

**sectend(X)** Returns the end address of sectionX.

**sectsize(X)** Returns the current (not final) size of sectionX.

**alignment(X)** Gives the alignment of previously defined symbolX. This value depends upon the base alignment of the section in whichX is defined, as follows:

*Word* aligned - value in range 0 -3

*Halford* aligned- value in range 0 -1

*Byte* aligned - value always 0

## Assembler Operators

The Assembler makes use of the following expression operators:

<i>Symbol</i>	<i>Type</i>	<i>Usage</i>	<i>Action</i>
( )	Primary	(a)	Brackets of Parenthesis
+	Unary	+a	a is positive
-	Unary	-a	a is negative (see Note <sup>1</sup> )
=	Binary	a=b	Assign or equate b to a
+	Binary	a+b	Increment a by b
-	Binary	a-b	Decrement a by b
*	Binary	a*b	Multiply a by b
/	Binary	a/b	Divide a by b, giving the quotient
%	Binary	a%b	Divide a by b, giving the modulus
<<	Binary	a<<b	Shift a to the left, b times
>>	Binary	a>>b	Shift a to the right, b times
~	Unary	~a	Logical compliment or NOT a
&	Binary	a&b	a is logically ANDed by b
^	Binary	a^b	a is exclusively ORed by b
!	Binary	a!b	a is inclusively ORed by b
	Binary	a b	Acts the same as db
<>	Binary	a<>b	a is unequal to b
<	Binary	a<b	a is less than b
>	Binary	a>b	a is greater than b
<=	Binary	a<=b	a is less than or equals b
>=	Binary	a>=b	a is greater than or equals b

### Note<sup>1</sup>

Since the Assembler will evaluate 32-bit expressions, the negation bit is Bit 31. Therefore, \$FFFFFF and \$FFFFFFF are positive hex numbers; \$FFFFFFFF is a negative number

### Note<sup>2</sup>

If a comparison evaluates as *true*, the result is returned as **-1**; if it evaluates as *false*, the result is returned as **0**.

## Hierarchy of Operators

Expressions in the Assembler are evaluated using the following precedence rules:

- Parentheses form the primary level of hierarchy and force precedence - their contents are performed first;
- Without the aid of parentheses, operators are performed in the order dictated by the hierarchy table;
- Operators with similar precedence are performed in the direction of their associativity - normally, from left to right, except unary operators.

<i>Operator</i>	<i>Direction</i>	<i>Description</i>
()	←	Primary
+, -, ~	→	Unary
<<, >>	→	Shift
&, !, ^	→	Logical
*, /, %	→	Multiplicative
+, -	→	Additive
>, <, <=, >=	→	Relational
=, <>	→	Equality

## RADIX

**Description** The **RADIX** directive is used to change the default base of integer values appearing in source code (base 10), by over-riding the default with whatever base you specify.

**Syntax**                      **RADIX**            *newbase*

**Examples**

```
var1            dw 17
radix            8
var2            dw 17
radix            16
var3            dw 17
var4            dw %1001
```

**Notes:**

var1 is a value of 17 (decimal)  
var2 contains a value of 15 (decimal)  
var3 contains a value of 23 (decimal)  
var4 contains a value of 17 (decimal)

**Remarks**

- Acceptable values for the new base are in the range of 2 to 16.
- Whatever the current default, the operand of the **RADIX** directive is evaluated to a decimal base.
- The **AN** assembler option (see chapter 9) will not be put into effect if the default **RADIX** is greater than 10, since the signifier **B** and **D** are used as digits in hexadecimal notation.

## ALIAS and DISABLE

**Description** These directives allow the programmer to avoid a conflict between the reserved system names of constants and functions and the programmer's own symbols. Symbols can be renamed by the **ALIAS** directive and the original names **DISABLE**'d, rendering them usable by the programmer.

**Syntax**

<i>newname</i>	<b>ALIAS</b>	<i>name</i>
	<b>DISABLE</b>	<i>name</i>

**Remarks** Symbolic names currently known to the Assembler may be **ALIAS**ed and **DISABLE**d. However, these directives must not be used to disable Assembler directives.

**Examples**

<code>_Offset</code>	<code>alias</code>	<code>offset</code>
	<code>disable</code>	<code>offset</code>
	<code>...</code>	
<code>_Offset</code>	<code>dh</code>	<code>_Offset(Lab)</code>
<code>offset</code>	<code>dh</code>	<code>*-pointer</code>



**CHAPTER 4****General Assembler Directives**

The Assembler provides a variety of functions and directives to control assembly of the source code and its layout in the target machine.

This chapter documents the Assembler directives which allow the programmer to control the processes of assembly, grouped as follows:

- **Assignment Directives**
- **Data Definition**
- **Controlling Program Execution**
- **Include Files**
- **Controlling Assembly**
- **Target-related Directives**

### **Assignment Directives**

The directives in this section are used to assign a value to a symbolic name. The value may be a constant, variable or string.

- **EQU**
- **SET** (and =)
- **EQUUS**
- **EQUR**
- **Rsize**
- **RSSET**
- **RSRESET**



## EQU

**Description** Assigns the result of the expression, as a constant value, to the preceding symbolic name.

**Syntax** *symbol name EQU expression*

**See Also** SET, EQU S

### Remarks

- The Assembler allows the assigned expression to contain forward references. If an **EQU** cannot be evaluated as it is currently defined, the expression will be saved and substituted in any future references to the equate (see Note below).
- It is possible to include an equate at assembly time, on the Assembler command line. This is useful for specifying major options of conditional assembly, such as *test mode* - see Assembler switches, chapter 2..
- Assigning a value to a symbol with **EQU** is absolute; an attempt at secondary assignment will produce an error. However, it is permissible to re-assign the *current* value to an existing symbol; typically, this occurs when subsidiary code redefines constants already used by the master segment.

**Examples**

```

Length    equ    4
Width     equ    8
Depth     equ    12
Volume    equ    Length*Width*Depth
DmaHigh   equ    $ffff8609
DmaMid    equ    DmaHigh+2

```

**Note** List equ Lastentry-Firstentry

if *Firstentry*, *Lastentry* not yet defined, then:

```
dw List+2
```

will be treated as

```
dw (Lastentry-Firstentry)+2
```

the equated expression is implicitly bracketed.

## SET

**Description** Assigns the result of the expression, as **variable**, to the preceding symbolic name.

**Syntax**        *symbol name* **SET**    *expression*  
                  *symbol name* =     *expression*

**See Also**     **EQU**

### Remarks

- **SET** and equals (=) are interchangeable
- Values assigned by a **SET** directive may be re-assigned at any time.
- The Assembler does not allow the assigned expression in **SET** directive to contain forward references. If **SET** cannot be evaluated as it is currently defined, an error is generated.
- If the symbol itself is used before it is defined, a warning is generated, and it is assigned the value determined by the preliminary pass of the Assembler.
- The symbol in **aSET** directive does not assume the type of the operand. It is, therefore, better suited to setting local values, such as in macros, rather than in code with a relative start position, such as a **SECTION** construct, which may cause an error - see *Examples*.

**Examples**

```
Loopcount    set        0
GrandTotal   =        SubTotalA+SubTotalB
xdim         set        Bsize<<SC
```

The following example will encrypt the string passed as the macro parameter.

```
cbb      macro    string
lc       =        0
          rept    strlen(\string)
cc       substr   lc+1,lc+1,\string; extract one
                                              character into
                                              label cc
          db      '\cc'^($A5+lc) ; encrypt the
                                              character stored
                                              in cc and define
                                              in memory
lc       =        lc+1 ; increment counter
          endr    ; do for all chars
          endm    in string
```

## EQU

**Description** Assigns a text or string variable to a symbol.

**Syntax**

```
symbol name EQU "text"
symbol name EQU 'text'
symbol name EQU symbol name
```

**See Also** EQU, SET

### Remarks

- Textual operands are delimited by double or single quotes. If it is required to include a double quote in the text string, delimit with single quotes or two double quotes; similarly, to include a single quote in the text, delimit with double quotes or two single quotes - see examples below.
- If delimiters are omitted, the Assembler assumes the operand to be the symbol name of a previously defined string variable, the value of which is assigned to the new symbol name.
- Point brackets, { and } are special delimiters used in Macros - see **MACRO** directive specification, chapter 5.
- Symbols equated with the **EQU** directive can appear at any point in the code, included as part of another text string. If there is the possibility of confusion with the surrounding text, a backslash (\) may be used before the symbol name and if necessary, after it, to ensure the expression is expanded correctly. See examples below.

### Examples

```
Singquote  equs 'What's the point? '
Singquote  equs "What's the point? "
```

```
Doubquote  equs "Say" "Hello" "and go"
Doubquote  equs 'Say "Hello" and go'
```

```
Program    equs "ABS v 1.2 "
Qtex       equs "What's the score? "
           db   "Remember to assemble
           \_filename ",0
```

```
Z          equs "123"  
          ----  
          dw   z + 4  
converts to  
          dw   123 + 4
```

whereas the following expression needs backslashes to be expanded correctly

```
          dw   number\z\a  
converts to  
          dw   number123a
```

```
SA          equs 'StartAddress '  
          ----  
          dw   \SA\4  
converts to  
          dw   StartAddress4
```

---

## EQR

**Description** Defines a symbol as an alternative for a register.

**Syntax** *symbol name* **EQR***register name*

**See Also** **REG**

**Remarks**

- The major use of the **EQR** directive is to improve the overall readability of the source code.
- In order that the Assembler can evaluate the expression correctly, dots are not allowed as part of the symbol name of a **EQR** (see example below).

**Examples** `lw t1,RGBinds(a2)`

This could be re-written using **EQR**'s, as follows:

```
Red      equr      t1
Green    equr      a2
        ...
lw      Red,RGBinds(Green)
```

## Rsize

**Description** Assigns the value of the `__RS` variable to the symbol, and advances the counter by the number of bytes, half-words or words, specified `count`.

**Syntax** `symbol name Rsize count`

where `size` is

<b>b</b>	byte (8 bits)
<b>h</b>	half word (16 bits)
<b>w</b>	word (32 bits)

(if `size` is not specified, `w` is assumed)

**See Also** **RSSET, RSRESET**

### Remarks

- This directive, together with the following two associated directives, operate on or with the Assembler variable, `__RS`, which contains the current offset.

### Examples

```

rsreset
Icon_no    rb    1
Dropcode   rh    1
Actcode    rh    1
Actname    rb   10
Objpos     rw    1
Artlen     rb    0

```

After each of the first five `RS` equates, the `__RS` pointer is advanced; the values for each equate are as follows:

Icon_no	0	(set to zero by RSRESET)
Dropcode	1	
Actcode	4	(Automatic Alignment set, advances the pointer to Alignment boundary)
Actname	6	
Objpos	16	
Artlen	20	

The last **rb** does not advance the **\_\_RS** pointer, since a count of zero is equivalent to an **EQUATE** to the **\_\_RS** variable.

---

## RSSET

**Description** Assigns the specified value to **\_\_RS** variable.

**Syntax**                      **RSSET**        *value*

**See Also**        **Rsize, RSRESET**

**Remarks**        This directive is normally used when the offsets are to start at a value other than zero.

**Examples**        See the *Rsize* directive



## RSRESET

**Description** Sets the `__RS` variable to zero.

**Syntax** `RSRESET [value]`

**See Also** *Rsize*, `RSSET`

**Remarks**

- Using this directive is the normal way to initialise the `RS` counter at the start of a new data structure.
- The optional parameter is provided for compatibility with other assemblers; if present, `RSRESET` behaves like the `RESET` directive.

**Examples** See the *Rsize* directive

## Data Definition

The directives in this section are used to define data and reserve space.

- **Dsize**
- *DCsize*
- **DSsize**
- **HEX**
- **DATA**
- **DATASIZE**
- **IEEE32**
- **IEEE64**

## *Dsize*

**Description** This directive evaluates the expressions in the operand field, and assigns the results to the preceding symbol, in the format specified by the size parameter. Argument expressions may be numeric values, strings or symbols.

**Syntax** *symbol name Dsize expression,...,expression*

where *.size* is **b** byte (8 bits)  
**h** half word  
**w** word

**See Also** *DCsize*

### Remarks

- Textual operands are delimited by double or single quotes. If it is required to include a double quote in the text string, delimit with single quotes or two double quotes; similarly, to include a single quote in the text, delimit with double quotes or two single quotes - see examples below. If delimiters are omitted, the Assembler assumes the operand to be the symbol name of a previously defined string variable, the value of which is assigned to the new symbol name.
- When the Automatic Alignment assembler option (/AE) is in force, directives for *half word* and *word* ensure that the program counter is aligned to the next word boundary.

**Examples**

Hexvals	dh	\$80d,\$a08,0,\$80d,0
Coords	dw	-15,46
Pointers	dw	StartMarker,EndMarker
ErrorMes	db	"File Error",0

**Notes** If the Assembler encounters a parameter that is out-of-range, an error is flagged; the following statements will produce errors:

db	257
db	-129
dh	66000
dh	-33000

## *DCsize*

**Description** This directive generates a block of memory of the specified length, containing the specified value.

**Syntax** **DCsize***length,value*

where *size* is **b** byte (8 bits)  
**h** half word (16 bits)  
**w** word

**See Also** *Dsize*

**Remarks** When the Automatic Alignment assembler option (/AE) is in force, DCB directives for half word and word ensure that the program counter is aligned to the next word boundary.

**Examples**

```
dcb 256,$7F ;generates 256 bytes containing $7F  
dcw 64,$12345678 ;generates 54 words containing  
$12345678
```

## Dsize

**Description** Allocates memory to the *symbol* of the specified *length* and initialises it to zero.

**Syntax** *symbol name* **D***size* *length*

where *size* is

<b>b</b>	byte (8 bits)
<b>h</b>	half word (16 bits)
<b>w</b>	word

**See Also** *Dsize*, *DCsize*

### Remarks

- When the Automatic Alignment assembler option (/AE) is in force **DS** directives for *word* and *long word* ensure that the program counter is aligned to the next *word* boundary.
- If this directive is used to allocate memory in **Group/Section** with the **BSS** attribute, the reserved area will not be initialised - see **Groups and Sections**, chapter 8.

**Examples** List            dsw            64

reserves an area 64 words long, and sets it to zero.

Buffer            dsb            1024

reserves a 1k bytes area, and sets it to zero.

## HEX

**Description** This directive takes a list of hex digits as an argument and assigns the resulting value to the preceding symbol. It is intended as a quick way of inputting small hex expressions.

**Syntax** *symbol name* **HEX** *hexlist*

**See Also** **INCBIN**

**Remarks** Data stored as **HEX** is difficult to read, less memory-efficient and causes more work for the Assembler. Therefore, it is suggested that the **HEX** statement is used for comparatively minor data definitions only. To load larger quantities of data, it is recommended that the data is stored in a file, to be **INCLUDED** as a binary file at runtime - see **Include Files**, chapter 4.

**Examples** HexStr     hex            100204FF0128

is another way of writing

HexStr     db            \$10,\$02,\$04,\$FF,\$01,\$28

## DATASIZE and DATA

**Description** Together, these directives allow the programmer to define values between 1 and 256 bytes long (8 to 2048 bits). The size of the **DATA** items must first be defined by a **DATASIZE** directive.

**Syntax**

```
DATASIZE size
DATA      value, value
```

where *value* is a numeric string, in hex or decimal, optionally preceded by a minus sign.

**See Also** **IEEE32, IEEE64**

**Remarks** If a *value* specified in the **DATA** directive converts to a value greater than can be held in *size* specified by **DATASIZE**, the Assembler flags an error.

**Examples**

```
datasize 8
...
data      $123456789ABCDEF0
data      -1,$FFFFFFFFFFFF
```

## IEEE32 and IEEE64

**Description** These directives allow 32 and 64 bit floating point numbers to be defined in **IEEE** format.

**Syntax**

```
IEEE32  fp.value
IEEE64  fp.value
```

**See Also** **DATA, DATASIZE**

**Examples**

```
ieee32    1.23,34e10
ieee64    123456.7654321e-2
```

## Controlling Program Execution

The directives in this section are used to alter the state of the program counter and control the execution of the Assembler.

- **ORG**
- **CNOP**
- **OBJ**
- **OBJEND**



## ORG

**Description** The **ORG** directive informs the Assembler of the location of the code in the target machine.

**Syntax** `ORG address[,parameter]`

where *address* is a previously-defined symbol, or a hex or decimal value, optionally preceded by a question mark (?) and followed by a (target-specific) numeric parameter.

**See Also** **OBJ, OBJEND, GROUP, SECTION**

### Remarks

- If a link file is output, the **ORG** directive must not be used - see **Groups and Sections**, chapter 8.
- If the program contains **SECTIONS**, a single **ORG** is allowed, and it must precede all **SECTION** directives. If the program does not utilise the **SECTION** construct, it may contain multiple **ORG**'s.

### Examples

```
Begin      org      $100
           move.w  sr, -(A7)

Program    equ      $4000
           ...
           org      Program
```

## CNOP

**Description** Resets the program counter to a specified *offset* from the specified *size* boundary.

**Syntax** `CNOP offset, size boundary`

**Remarks** In code containing **SECTIONS**, the Assembler does not allow the program counter to be reset to a size boundary greater than the alignment already set for that section. Therefore, a **CNOP** statement with a size boundary of 2 is not allowed in a section that is byte-aligned.

**Examples**

```
section prime
Firstoff = 512
Firstsize = 2
...
cnop Firstoff, Firstsize
```

sets the program counter to 512 bytes above the next word boundary.

## OBJ and OBJEND

**Description** **OBJ** forces the code following it to be assembled as if it were at the specified address, although it will still appear following on from the previous code.

**OBJEND** terminates this process and returns to the **ORG'd** address value.

**Syntax**                      **OBJ**                      *address*

**OBJEND**

**See Also**                    **ORG**

### Remarks

- The **OBJ - OBJEND** construct is useful for code that must be assembled at one address (for instance, in a ROM cartridge), but will be run at a different address, after being copied there.
- Code blocks delimited by **OBJ - OBJEND** cannot be nested.

### Examples

```
org      $100
dw      *
dw      *

obj      $200
dw      *
dw      *

objend
dw      *
dw      *
```

The above code will generate the following sequence of words, starting at address \$100:

```
$100
$104
$200
$204
$110
$114
```

## Include Files

The source code for most non-trivial programs is too large to be handled as a single file. It is normal for a program to be constructed of subsidiary files, which are called together during the assembly process.

The directives in this section are used to collect together the separate source files and control their usage; also discussed are operators to aid the control of code to be assembled from **INCLUDEd** files.

- **INCLUDE**
- **INCBIN**
- **DEF**
- **REF**

## INCLUDE

**Description** As the source code for most no-trivial programs is too large to be handled as a single file, it is normal for a program to be constructed of subsidiary files, which are called together during the assembly process. This directive tells the Assembler to draw in and process another source file, before resuming the processing of the current file.

**Syntax** `INCLUDE filename`

where *filename* is the name of the source file to be processed, including drive and path identifiers - see **Note**. The *filename* may be surrounded by quotes, but they will be ignored.

**See Also** `INCBIN`

### Remarks

Traditionally, there will be one main file of source code, which contains `INCLUDE`'s for all the other files.

The `/j` switch can be used to specify a search path for `INCLUDE`d files - see **Assembler Options** chapter 9.

**Examples** A typical start to a program may be:

```
codestart    section    short1
             jmp        entrypoint

             db         _hours,_minutes
             db         _day,_month
             dh         _year

             include   vars1.s

             section   short2

             include   vars2.s

             section   code

             include   graph1.s
             include   graph2.s
```

```
        include    maths.s
        include    trees.s
        include    tactics.s

entrypoint li      t0,8
          lw      a0,fred
          ...
```

**Note:** Since a path name contains backslashes, the text in the operand of an INCLUDE statement may be confused with the usage of text previously defined by an EQU directive. To avoid this, a second backslash may be used or the backslash may be replaced by a forward slash.

## INCBIN

**Description** Informs the Assembler to draw in and process binary data held in another source file, before resuming the processing of the current file.

**Syntax** *symbol* **INCBIN** *filename[,start,length]*

- *filename* is the name of the source file to be processed, including drive and path identifiers. Optionally, the filename may be surrounded by quotes, which will be ignored;
- *start* and *length* are optional values, allowing selected portions of the specified file to be included.

**See Also** **INCLUDE, HEX**

### Remarks

- This directive allows quantities of binary data to be maintained in a separate file and pulled into the main program at assembly time; typically, such data might be character movement strings, or location co-ordinates. The Assembler is passed no information concerning the type and layout of the incoming data. Therefore, labeling and modifying the **INCBIN**ed data is the responsibility of the programmer.
- The **/j** switch can be used to specify a search path for **INCLUDE**ed files - see Assembler Options, chapter 9.

**Examples** Charmove incbin d:\source\charmov.bin

**Note:** Since a path name contains backslashes, the text in the operand of an **INCBIN** statement may be confused with the usage of text previously defined by an **EQU**S directive. To avoid this, a second backslash may be used or the backslash may be replaced by a forward slash.

Thus, include f:\source\charmov.bin

may be re-written as

include d:\\source\\charmov.bin or

include d:/source/charmov.bin

**Note** The nominated file may be accessed selectively, by specifying a position in the file, from which to start reading, and **length**. **Note that:**

- if *start* is omitted, the INCBIN commences at the beginning of the file;
- if the *length* is omitted, the INCBIN continues to the end of the file;
- if both *start* and *length* are omitted, the entire file is INCBINed.

## REF

**Description** **REF** is a special operator to allow the programmer to determine which segments of code are to be **INCLUDEd**.

**Syntax** `[~]REF(symbol)`

The optional preceding tilde (~) is synonymous with **NOT**.

**Remarks** **REF** is *true* if a reference has previously been encountered for the symbol in the brackets.

**Examples**

```

Links      if          ref(Links)
           lw          r1,8(a0)
           ...
           jr          ra
           endif

```

The *Links* routine will be assembled if a reference to it has already been encountered.



## DEF

**Description** Like the **REF** operator, **DEF** is a special function. It allows the programmer to determine which segments of code have already been **INCLUDEd**.

**Syntax**                            [~]DEF(symbol)

The optional preceding tilde (~) is synonymous with **NOT**.

**Remarks**    **DEF** is *true* if the symbol in the brackets has previously been defined.

**Examples**

```
if ~def(loadadr)
loadadr equ $1000
execadr equ $1000
relocadr equ $80000-$300
endc
```

The address equates will be assembled if load\_addr has not already been defined.

## Controlling Assembly

The following directives give instructions to the Assembler during the assembly process. They allow the programmer to select and repeat sections of code:

- **END**
- **IF**
- **ELSE**
- **ELSEIF**
- **ENDIF**
- **CASE**
- **ENDCASE**
- **REPT**
- **ENDR**
- **WHILE**
- **ENDW**
- **DO**
- **UNTIL**

## END

**Description** The **END** directive informs the Assembler to cease its assembly of the source code.

**Syntax**                                **END** [address]

**See Also**        **REGS**

**Remarks**

- The inclusion of this directive is mostly cosmetic, since the Assembler will cease processing when the input source code is exhausted.
- The optional parameter specifies an initial address for the program. See also the **REGS** statement, in the section **-Target-Related Directives**, chapter 4.

**Example**        startrel    lw            t0,(a0)  
                              addu          r3,r2,r1  
                              ...  
                              jr            ra  
                              end

---

## IF, ELSE, ELSEIF, ENDIF, ENDC

**Description** These conditional directives allow the programmer to select code for assembly.

**Syntax**                                **IF**            [~]expression  
   **ELSE**  
   **ELSEIF**        [~]expression  
   **ENDIF**  
   **ENDC**

**See Also**        **CASE**

**Remarks**

- The **ENDC** and **ENDIF** directives are interchangeable.
- If the **ELSEIF** directive is used without a following expression, it acts the same as an **ELSE** directive.
- The optional tilde, preceding the operand expression, is synonymous with **NOT**. Its use normally necessitates the prudent use of brackets to preserve the sense of expression.

**Examples**

```

sec_dir    if          Nintendo 1
           equ         2

```

```

sec_dir    elseif     Target
           equ         1

```

```

sec_dir    else
           equ         3
           endif

```

```

round      if          ~usesquare
           macro
           addu        \1,\2,\3
           endm

```

```

round      elseif
           macro
           endm
           endc

```

```

ldimm     macro        dest,imm
           if          (\imm>-
                       32768)&(\imm<32768 )
           addiu       \dest,r0,\imm
           else        lui \dest,(\imm)>>16
           if          (\imm)&$ffff
           ori         \dest,\det,(\imm)&$ffff
           endif
           endif
           endm

```

## CASE and ENDCASE

**Description** The **CASE** directive is used to select code in a multiple-choice situation. The **CASE** argument defines the expression to be evaluated; if the argument(s) after the *equals sign* are *true*, the code that follows is assembled. The *equals-questionmark* case is selected if no previous case is *true*.

**Syntax**

```

CASE      expression
=expression[,expression]
=?
ENDCASE

```

**See Also** **IF conditionals**

**Remarks** In the absence of *equals-question mark*(=?) case, if the existing cases are unsuccessful, the case-defined code is not assembled.

**Examples** The following is an alternative for the example listed under the **IF** directive - see chapter 4.

```

Target      equ   Nintendo1
            ...

            case Target

=Nintendo1
sec_dir     equ   2

=Nintendo2
sec_dir     equ   1

=?         db    "New Version",0
sec_dir     equ   3

            endcase

```

## REPT, ENDR

**Description** These directives allow the programmer to repeat the code between the **REPT** and **ENDR** statements. The number of repetitions is determined by the value of *count*.

**Syntax**

```
REPT      count
...
ENDR
```

**See Also** **DO, WHILE**

**Remarks** When used in a Macro, **REPT** is frequently associated with the **NARG** function.

**Examples**

```
rept      12
dh        0,0,0,0
endr
```

---

```
cbb      macro      string
lc        =          0
rept      strlen(\string)
cc        substr    lc+1,lc+1,\string
db        "\cc"^( $A5+lc)
lc        =          lc+1
endr
endm
```

## WHILE, ENDW

**Description** These directives allow the programmer to repeat the code between the WHILE and ENDW statements, as long as the expression in the operand holds *true*.

**Syntax**

```

WHILE    expression
...
ENDW

```

**See Also** REPT, DO

**Remarks** Currently, any string equate substitutions in the **WHILE** expression take place once only, when the **WHILE** loop is first encountered - see **Note** below for the ramifications of this.

**Examples**

```

MultP    equ    16
...
Indic    =      MultP
while    Indic > 1
lw      r1, term(a0)
...
Indic    =      Indic-1
endw

```

**Note:** Because string equates are only evaluated at the start of the **WHILE** loop, the following will not work:

```

s        equ    "x"
while    strlen("\s") < 4
dcb     "\s", 0
s        equ    "\s\x"
endw

```

To avoid this, set a variable each time round the loop to indicate that looping should continue:

```

s      equs      "x"
looping =      -1
      while     looping
      db       "\s",0
s      equs      "\s\x"
looping =      strlen("\s") < 4
      endw

```

## DO, UNTIL

**Description** These directives allow the programmer to repeat the code between the **DO** and **UNTIL** statements, until the specified expression becomes *true*.

**Syntax**

```

DO
...
UNTIL expression

```

**See Also** **REPT, WHILE**

**Remarks** Unlike the **WHILE** directive, string equates in a **UNTIL** expression will be re-evaluated each time round the loop.

**Examples**

```

MultP      equ      16
...
Indic      =      MultP
do
lw        r1,term(a0)
...
Indic      =      Indic-1
until     Indic<=1

```



### Target-Related Directive

The following directive allows the programmer to specify certain initial parameters in the target machine:

- **REGS**

## REGS

**Description** This directive allows the programmer to specify certain initial parameters in the target machine. If a *CPE* file is produced or object code output is directed to the target, the **REGS** directive specifies the contents of the registers at the start of code execution.

**Syntax** **REGS** *regcode=expression[,regcode=expression]*

where *regcode* is the mnemonic name of a register, such as **as1**, **PC**.

**Remarks** This directive is not available for relocatable code which is specific to the target or pure binary code.

**Example** `regs pc = __SN_ENTRY_POINT`

Register assigns can be declared on one line, separated by commas.

**CHAPTER 5****Macros**

The Assembler provides extensive macro facilities; these allow the programmer to assign names to complete code sequences. They may then be used in the main program like existing assembler directives.

This chapter discusses the following topics, directives and functions:

- **MACRO, ENDM**
- **MEXIT**
- **Macro Parameters**
- **SHIFT, NARG**
- **MACROS**
- **PUSHP, POPP**
- **PURGE**
- **TYPE**

## MACRO, ENDM, MEXIT

The Assembler provides extensive macro facilities; these allow the programmer to assign names to complete code sequences. They may then be used in the main program like existing assembler directives.

**Description** A *macro* consists of the source lines and parameter place markers between the **MACRO** directive and the **ENDM**. The label field is the symbolic name by which the macro is invoked; the operand allows the entry of a string of parameter data names.

When the assembler encounters a directive consisting of the *label* and optional parameters, the source lines are pulled into the main program and expanded by substituting the place markers with the invocation parameters. The expansion of the macro is stopped immediately if the assembler encounters **MEXIT** directive.

**Syntax**

```

Label      MACRO      [symbol,..symbol]
           ...
           MEXIT
           ...
           ENDM

```

**See Also**    **MACROS**

### Remarks

- Note that the invocation parameter string effectively starts at the character after the macro name, that is, the *dot* (.) character. Text strings, as well as **b**, **.w** and **.l** are permissible parameters - see **Parameters** below.
- Control structures within macros must be complete. Structures started in the macro must finish before the **ENDM**; similarly, a structure started externally must not be terminated within the macro. To imitate a simple control structure from another assembler, a short macro might be used - see **MACROS** below.

**Examples**

```

remove      macro
             dh          -2,0,0
             endm

```

---

```

Form       macro
             if          strcmp('\1','0')
             dh          0
             else
             dh          \1-Form Base
             endif
             endm

```

## Macro Parameters

**Parameters** Macro parameters obey the following rules:

- The parameters listed on the macro invocation line may appear at any point in the code declared between the **MACRO** and **ENDM** statements. Each parameter is introduced by *abackslash*(\); where this may be confused with text from an **EQU**, a backslash may also follow the parameter.
- Up to thirty two different parameters are allowed, numbered **0** to **31**. **\0** is a special parameter which gives the contents of the *size field* of the macro directive when it was invoked, that is, the text after the point symbol (.) This includes not only **.b**, **.h** or **.w**, but also any text:

**Example**

```
zed      macro
          \0
          endm
          ...
          zed.nop
```

will generate a NOP instruction.

Instead of the **\0** to **\31** format, parameters can be given symbolic names, by their inclusion as operands to the **MACRO** directive. The preceding *backslash*(\) is not mandatory; however, if there is the possibility of confusion with the surrounding text, a *backslash* may be used before and after the symbol name to ensure the expression is expanded correctly:

**Example**

```
Position macro      A,B,C,Pos,Time
          dh          \Time*(\A*\Pos+\B*\Pos+\C*\Pos)
          endm
```

Surrounding the operand of an invoked macro with *greater than* and *less than* signs (<...>), allows the use of comma and space characters.

```

Example Credits macro
           dh      \1,\2
           db      \3
           db      0
           even
           endm
           ...
Credits 11,10,<A BS, from Jones>

```

- Continuation Lines- when invoking a macro, it is possible that the parameter list will become overlong. As with any directive statement, the line can be terminated by an *ampersand*(&) and continued on the next line to improve readability.

```

Example chstr macro
           rept    narg
           db      k_\1
           shift
           endr
           db      0
           even
           endm
           ...
cheatstr chstr i,c,a,n,b,a,r,e,l,y,&
           s,t,a,n,d,i,t

```

## Special Parameters

There are a number of special parameter formats available in macros, as follows:

### Converting Integers to Text

The parameters `\#` and `\$` replace the decimal (`\#`) or hex (`\$`) value of the symbol following them, with their character representation. Commonly, this technique is used to access *Run Date* and *Time*:

#### Example

```
RunTime    db          "\#_hours:\#_minutes:&
              \#_seconds"
```

this expands to the form hh:mm:ss, as follows

```
RunTime    db          "21:08:49"
```

### Generating Unique Labels

The parameter `\@` can be used as the last characters of a label name in a macro. When the macro is invoked, this will be expanded to an underscore followed by a decimal number; this number is increased on each subsequent invocation to give a unique label.

#### Example

```
Slots      macro
            add   r1,r0,r0
            lw    r1,\1
            beq   r11,r0,1
next\@     addiu   f3,r0,1

            bgt
dun\@
            endm
            ...
Slots      freeobl,numslot1
```

Each time the *Slots* macro is used, new labels in the form `next_001` and `dun_001` will be generated.

## Entire Parameter

If the special parameter `\_` (backslash underscore) is encountered in a macro, it is expanded to the complete argument specified on the macro invocation statement.

<b>Examples</b>	All	macro	
		db	\_
		endm	
	...		
	All		1, 2, 3, 4

*will generate*

db	1, 2, 3, 4
----	------------

## Control Characters

The parameter `\^x`, where *x* denotes a control character, will generate the specified control character.

## Using the Macro Label

The label heading the invocation line can be used in the macro, by specifying the first name in the symbol list of the **MACRO** directive to be an asterisk (\*), and substituting `\*` for the label itself. However, the resultant label is not defined at the current program location. Therefore, the label remains *undefined* unless the programmer gives it a value.

## Extended Parameters

The Assembler accepts a set of elements, enclosed in curly brackets `{ }` to be passed to a macro parameter. The **NARG** function and **SHIFT** directive can then be used to handle the list:

<b>Example</b>	cmd	macro	
	cc	eus	{\1}
		rept	narg(cc)
		\cc	
		shift	cc
		endr	
		endm	
		jr	ra



## SHIFT, NARG

**Description** These directives cater for a macro having a variable parameter list as its operand. The **NARG** symbol is the number of arguments on the macro invocation line; **SHIFT** directive shifts all the arguments one place to the left, losing the leftmost argument.

**Syntax**     *directive*     **NARG**  
                                   ...  
                                   **SHIFT**

where **NARG** is a reserved, predefined symbol.

**See Also**     **Extended Parameters**

**Examples**

```

routes    macro
           rept      narg
           if        strcmp('\1','0')
           dh        0
           else
           dh        \1-routebase
           endif
           shift
           endr
           endm
           ...
routes    0, gosouth_1

```

This example goes through the list of parameters given to the macro and defines a half word of \$0000 if the argument is zero or a 16 bit offset into the 'routebase' table of the given label.

## MACROS

**Description** The **MACROS** directive allows the entry of a single line of code as a macro, with no associated **ENDM** directive. The single line of code can be a control structure directive.

**Syntax**     *Label*           **MACROS**    [*symbol,..symbol*]

**See Also**    **MACRO**

**Remarks**    The **MACROS** directive may be used to stand in for a single, complex code line. Often, the short macro allows the programmer to synthesise a directive from another assembler. Including the **k** option on the command line will cause several macros emulating foreign directives to be generated.

**Examples**

```

a            =            0        ;a=0 do explosion
                          ;a=1 do offset calculation

              if            0
boom         macros
              jal            explos\1
              else
boom         macros
              lw            r1,blow up-tactbase(\1
              endif

```

## PUSHP, POPP

**Description** These directives allow text to be pushed into, and then popped from, a string variable.

**Syntax**

<b>PUSHP</b>	<i>string</i>
<b>POPP</b>	<i>string</i>

**Remarks** There is no requirement for the **PUSH** and corresponding **POPP** directives to appear in the same macro.

**Examples**

```
makeframe      macro      stksize
                pushp      \stksize
                sub        sp,\stksize\1
                endm

freeframe      macro
                popp        stksize
                add        sp,\stksize
                endm
```

This means the user does not have to specify `stksize` when using `freeframe`. The user must ensure that calls to `makeframe` and `freeframe` are balanced.

## PURGE

**Description** The **PURGE** directive removes an expanded macro from the internal tables and releases the memory it occupied.

**Syntax**                                **PURGE**     *macroname*

**Remarks**     It you need to redefine a macro, it is not necessary to purge it first as this is done by the Assembler.

**Examples**     HugeM     macro  
                              dh            \1  
                              dh            \2  
                              ...  
                              dh            \31  
                              endm  
  
                              HugeM        para1,103,faultlevel,&  
                              ...  
  40,50,para31  
  
                              purge        HugeM

## TYPE

**Description** **TYPE** is a function used to provide information about a symbol. It is frequently used with a macro to determine the nature of its parameters. The value is returned as a word; the meanings of the bit settings are given below.

**Syntax** **TYPE**(*symbol*)

The reply word can be interpreted as follows:

<b>Bit 0</b>	Symbol has an absolute value
<b>Bit 1</b>	Symbol is relative to the start of the Section
<b>Bit 2</b>	Symbol was defined using <b>SET</b>
<b>Bit 3</b>	Symbol is a Macro
<b>Bit 4</b>	Symbol is a String Equate ( <b>EQU</b> )
<b>Bit 5</b>	Symbol was defined using <b>EQU</b>
<b>Bit 6</b>	Symbol appeared in an <b>XREF</b> statement
<b>Bit 7</b>	Symbol appeared in an <b>XDEF</b> statement
<b>Bit 8</b>	Symbol is a Function
<b>Bit 9</b>	Symbol is a Group Name
<b>Bit 10</b>	Symbol is a Macro parameter
<b>Bit 11</b>	Symbol is a short Macro ( <b>MACROS</b> )
<b>Bit 12</b>	Symbol is a Section Name
<b>Bit 14</b>	Symbol is a Register Equate ( <b>EQUR</b> )



**CHAPTER 6****String Manipulation Functions**

To enhance the Macro structure, the Assembler includes powerful functions for string manipulation. These enable the programmer to compare strings, examine text and prepare subsets.

This chapter covers the following string handling functions and directive:

- **STRLEN**
- **STRCMP**
- **INSTR**
- **SUBSTR**

## STRLEN

**Description** A function which returns the length of the text specified in the brackets.

**Syntax** `STRLEN(string)`

**See Also** `STRCMP`

**Remarks** The `STRLEN` function is available at any point in the operand.

**Examples**

```
Nummov      macro
              rept      strlen(\1)
              lw        r3,(a0
              add       a0,4
              sw        r3,(a1)
              add       a1,4
              endr
              endm
              . . .
Nummov      12345
```

The number of characters in the string is used as the extent of the loop.



## STRCMP

**Description** A function which compares two text strings in the brackets, and returns *true* if they match, otherwise it returns *false*.

**Syntax** `STRCMP(string1,string2)`

**See Also** `STRLEN`

**Remarks** When comparing two text strings, the `STRCMP` function starts numbering the characters in the target texts from one.

**Examples**

```
Vers      equ      "Acs"
...
if        strcmp("\Vers","Sales")
lh        v0,Sa1Ind (a0)
else
  if      strcmp("\Vers","Acs")
  lh      v0, AcInd (a0)
  else
    if    strcmp("\Vers","Test")
    lh    v0, TstInd (a0)
  endif
endif
endif
```



## SUBSTR

**Description** This directive assigns a value to a symbol; the value is a sub-string of a previously specified text string, defined by the *start* and *end* parameters. The *start* and *end* parameters will default to the start and end of the string, if omitted.

**Syntax**     *symbol*     **SUBSTR**     [*start*],[*end*],*string*

**See Also**     **INSTR, EQU**

**Remarks**     When assigning a sub-string to a symbol, the **SUBSTR** directive starts numbering the characters in the source text from one.

**Examples**

```

Message    equ      "A short Sample String"
Part1      substr   9,14,"\Message"
Part2      substr   16,, "\Message"
Part3      substr   ,7, "\Message"
Part4      substr   ,, "\Message"

```

where Part1 equals *Sample*  
 Part2 equals *String*  
 Part3 equals *A short*

The last statement is equivalent to an **EQU** assigning the whole of the original string to Part4.

```

Cbb        macro    string
cc         =        0
           rept     strlen(\string)
cc         substr   lc+1,lc+1,\string
           db       '\cc'^( $A5+1c)
           lc+1
           endr
           endm

```

Again, this is an example of encryption of a string.



**CHAPTER 7****Local Labels**

As a program develops, finding label names that are both unique and definitive becomes increasingly difficult. Local Labels ease this situation by allowing meaningful label names to be re-used.

This chapter covers the following topics and directives:

- **Local Label Syntax and Scope**
- **MODULE and MODEND**
- **LOCAL**

## Syntax and Scope

### Syntax

- Local Labels are preceded by a local label signifier. By default, this is `@`; however, any other character may be declared by using the `l` option in an `OPT` directive or on the Assembler command line - see **Assembler Options** chapter 9.
- Local label names follow the general label rules, as specified in chapter 3.
- Local labels are not de-scoped by the expansion of a macro.

### Scope

The region of code within which a Local Label is effective is called *scope*. Outside this area, the label name can be re-used. There are three methods of defining the scope of a Local Label:

- The scope of a local label is implicitly defined between two non-local labels. Setting a variable, defining an equate or RS value does not de-scope current local labels, unless the `d` option has been used in an `OPT` directive or on the Assembler command line - see **Assembler Options**, chapter 9..
- The scope of a Local Label can also, and more normally, be defined by the directives `MODULE` and `MODEND` - see chapter 7.
- To define labels (or any other symbol type) for local use in a macro, the `LOCAL` directive can be used - see chapter 7..

### Examples

```

plot2      lb      t0,loc(t1)
           nop
           subiu   t0,t2
           bne    zero,t0,@chk1
           nop
           jal    lcolor
           nop
           b      setplot
           nop

@chk1
Setplot   set      *
Plot3     ---
           b      @chk1
           nop

```

## MODULE and MODEND

**Description** Code occurring after a **MODULE** statement, and up to and including the **MODEND** statement, is considered to be a *module*. Local labels defined in a *module* can be re-used, but cannot be referenced outside the *module's* scope. A Local label defined elsewhere cannot be referenced within the *current module*

**Syntax**

```

MODULE
...
...
MODEND

```

**See Also** **LOCAL**

### Remarks

- Modules can be nested.
- The **MODULE** statement itself is effectively a non-local label and will de-scope any currently active default scoping.
- Macros can contain modules or be contained in a *module*. A local label occurring in a module can be referred to by a macro residing anywhere within the module. A module contained within a macro can effectively provide labels local to that macro.

**Examples**

```

mystrcmp  module
           move      t2,a0
           move      v0,zero
@lp       lbu        t0,0(t2)
           addi      t2,t2,1
           lbu       t1,0(a2)
           addiu     a2,a2,1
           bne      t0,t1,@diff
           nop
           bnez     t0,@lp
           nop
           li       v0,1           ;true
@diff     jr        ra
           nop
           modend   ;mystrcmp

```

## LOCAL

**Description** The **LOCAL** directive is used to declare a set of macro-specific labels.

**Syntax**                                **LOCAL**     *symbol,...,symbol*

**See Also**     **MODULE**

**Remarks**

- The scope of symbols declared using the **LOCAL** directive is restricted to the host macro.
- The **LOCAL** directive does not force a type on the symbol set that makes up its operand. In practice, therefore, such symbols can be used as equates, string equates or any other type, as well as labels.

**Examples**

```
print        macro        string
             local        mylabel
             jal            stringprint
             nop
             db            \string,0
mylabel     cnop          0,4
             endm
```



## CHAPTER 8

## Structuring the Program

Normally, the organisation of the memory of the target machine does not match the layout of the source files. The Assembler however, uses Groups and Sections to create a structured target memory **and** relocatable program sections.

This chapter covers the following topics and directives:

- **SECTION**
- **GROUP**
- **PUSHS and POPS**
- **SECT and OFFSET**

## GROUP

**Description** This directive declares a group with up to seven group attributes.

**Syntax** *GroupName* **GROUP** [*Attribute,..Attribute*]

where an attribute is one of the following - see below for descriptions:

**WORD**  
**BSS**  
**ORG**(*address*)  
**FILE**(*filename*)  
**OBJ**(*address*)  
**SIZE**(*size*)  
**OVER**(*GroupName*)

**See Also** SECTION

**Remarks** Group Attributes are interpreted as follows:

**WORD** - the *group* may be accessed using absolute word addressing. Note that this will only have an effect if the **ow+** parameter has been used to allow optimisation to occur.

**Example** Group1 group word

**BSS** - no initialised data to be declared in this *group*.

**Example** Group1 group bss

**ORG** - sets the **ORG** address of the *group*, without reference to the other *group* addresses. If this attribute is omitted, the *group* will be placed in memory, following on from the end of the previous *group*.

**Example**

```

org      $100
G1      group
G2      group      org($400)
G3      group

```

will place the groups in the sequence G1,G2,G3

**FILE** - outputs a *group*, such as an overlay, to a its own binary file; other groups will be output to the declared file.

**Example**

```

Group1  group      org($400),file("charov.bin")

```

**OBJ** - sets the group's **OBJ** address. Code is assembled as if it is running at the **OBJ** address but is placed at the group's **ORG** address. If no address is specified then the **OBJ** value is the same as the group's **ORG** address.

**Examples**

```

Group1  group      org($400),obj($1000)
Group2  group      org($800),obj()

```

**SIZE** - specifies the maximum allowable size of the *group*. If the size exceeds the specified size, the assembler reports an error.

**Example**

```

Group1  group      size(32768)

```

**OVER** - overlays this *group* on the specified *group*. Code at the start of the second group is assembled at the same address as the start of the first group. The largest of the overlaid groups' sizes is used as the size of each group.

**Note:** It is necessary to use the **FILE** attribute to force different overlays to be written to different output files.

**Example**

```

Group2  group      over(Group1)

```

## SECTION

**Description** This directive declares a logical code section.

**Syntax** `SECTIONsize SectionName[,Group]`

*SectionName* `SECTIONsize [Attribute,..Attribute]`

The second format is a special case, designed to allow definition of a section with group attributes - see below for a description.

**See Also** **GROUP**

### Remarks

- Unless the section has been previously assigned, the section will be placed in an unnamed default group, if the **GROUP** name is omitted
- It is possible to define a section with group attributes. The assembler will automatically create a group with the section name preceded by a tilde (~) and place the section in it.

**Example** `Sect1        section    bss`

defines *Sect1*, with the **BSS** attribute, in a group called *~Sect1*.

- The *size* parameter can be **b**, **h** or **w**; if the parameter is omitted, the default size is **word**. When a size is specified on a section directive, alignment at that size is forced *at that point*. The start of the section is aligned on a boundary based on the largest size on any of the entries to that section - in all modules in the case of linked code.

**Note:** If a section is sized as byte, you cannot use the **EVEN** directive in the section. Furthermore, the **CNOP** directive cannot be used to re-align the Program Counter to a value greater than the alignment of the host section - See chapter 4.

- If sections are used to structure application code, only a single **ORG** directive can be used; this must precede all section definitions. Groups and Sections may have **ORG** attributes to position them.

No **ORG** directives or attributes are permitted when producing linkable output. Within a group, sections are ordered in the sequence that the Linker encounters the section definitions.

### Example

```
Sectionb  one
db        1,2,3

section   two
db        10,11,12

sectionb  one
db        4,5

section   two
db        13,14
```

Will produce the following sections and bytes:

```
one      1,2,3,4,5
two      10,11,12,0,13,14
```

## PUSHS and POPS

**Description** These directives allow the programmer to open a new, temporary section then return to the original section. **PUSHS** saves the current section, **POPS** restores it.

**Syntax****PUSHS****POPS**

**Examples**

```
plotcomp    lw      t0,8(t1)
            ---
passdl      equ     *
            pushes
            section dolight
            dw      passdl
            ...
            pops
```

This example shows **PUSHS** and **POPS** being used to pass system information between sections, in the form of the location counter.

## SECT and OFFSET

**Description** The **SECT** function returns the address of the section in which the symbol in the brackets is defined. The **OFFSET** function returns the offset value from the beginning of the section.

**Syntax**

<b>SECT</b>	<i>(expression)</i>
<b>OFFSET</b>	<i>(expression)</i>

### Remarks

- If a link is being performed, the **SECT** function is evaluated when it is linked; if there is no link it will be evaluated when the second pass has finished.
- Likewise, if a link is being performed, the **OFFSET** function is evaluated when it is linked; however, if there is no link the **OFFSET** will be evaluated during the first pass.

**Examples**

dh	sect(Table1)
dh	sect(Table2)
dh	offset(*)





**CHAPTER 9****Options, Listings and Errors**

This chapter completes the discussion of the Assembler and its facilities. It covers methods of determining run-time Assembler options, producing listings and error-handling, as well as passing information to the Linker:

- **OPT**
- **Assembler Options**
- **PUSHO and POPO**
- **LIST and NOLIST**
- **INFORM**
- **FAIL**
- **XREF, XDEF and PUBLIC**
- **GLOBAL**

## OPT

**Description** This directive allows Assembler options to be enabled or disabled in the application code. See 'Assembler Options' below for a full listing.

**Syntax** `OPT option,..option`

**See Also** **PUSHO, POPO**

**Remarks**

- An option is turned on and off by the character following the option code:
  - + (plus sign) = ON
  - (minus sign) = OFF
- Options may also be enabled or disabled by using the **thO** switch on the Assembler command line - see Command Line Syntax, chapter 2.

**Examples** `opt an+,l:,e-`

## Assembler Options

The following reference list shows the default settings for the various options and optimisations available during assembly. More detailed descriptions are given below.

<i>Option</i>	<i>Description</i>	<i>Default</i>
<b>AE</b>	Enable Automatic Alignment Mode	On
<b>AN</b>	Enable Alternate Numeric mode	Off
<b>AT</b>	Allow Assembler to use temporary Register	On
<b>C</b>	Activate/ Suppress Case sensitivity	Off
<b>D</b>	Allow <b>EQU</b> or <b>SET</b> to descope local labels	Off
<b>E</b>	Print lines containing errors	On
<b>H</b>	Automatic hazard removal (insert NOP)	Off
<b>Lx</b>	Substitute <i>x</i> for Local Label signifier	Off
<b>M</b>	Enable/disable macro instructions	On
<b>N</b>	Insert NOP in branch delay slots	Off
<b>R</b>	Allow \$ prefixed register names	Off
<b>S</b>	Handle equated names as labels	Off
<b>T</b>	Automatically truncate values in db/dh/dw statements	Off
<b>V</b>	Write Local Labels to symbol file	Off
<b>W</b>	Print warning messages	On
<b>WS</b>	Operands may contain white space	Off
<b>X</b>	Assume <b>XREFs</b> in defined section	Off

## Option Descriptions

### AE - Automatic Alignment

When using the word and long word forms **DC**, **DCB**, **DS** and **RS**, enabling this option forces the program counter to the following boundary prior to execution. The default setting for this option is **AE+**.

### AN - Alternate Numeric

The default setting for this option is **AN-** but setting it to **AN+** allows the inclusion of numeric constants in Zilog or Intel format, i.e. followed by **H**, **D** or **B** to signify **Hex**, **Decimal** or **Binary**, e.g. **0F123H = \$F123**. See also the section on the **RADIX** directive - chapter 3.

### AT - Alternate Assembler to use Temporary Register

Within MIPS standard format code, some instructions are not actually instructions but macros, for example:

```
sw t0,fred
```

when assembled will produce the following:

```
Lui      at,fred>>16
sw      t0,fred& $ffff(at)
```

This trashes the AT Register.

Warnings will be generated when the AT Register is used if this option is set **AT+** but **errors** will be inserted if it is set to **AT-**.

The default setting is **AT+**.

### C - Case Sensitivity

When this option is set to **C+**, the case of the letters in a label's name is significant; for instance, **SHOWSTATS**, **ShowStats** and **showstats** would all be legal. The default setting is **C-**.

### D - Descope Local Labels

The default setting for this option is **D-** but if it is set to **D+**, local labels will be descope if an **EQU** or **SET** directive is encountered.

### E - Error Text Printing

If this option is enabled, the text of the line which caused an Assembler error will be printed together with the host file name and line number. The default setting for this option is **E+**.

**H - Automatic Hazard Removal**

If this option is set to the default **H-**, the Assembler will warn the user of any pipeline hazards. If it is set to **H+**, it will insert a NOP into the code. Note that this does not apply to instructions following returns at the end of subroutines.

**L- Local Label Signifier**

Local labels are signified by a preceding AT sign (@). This option allows the use of the character following the option letter as the signifier. Thus, **L+** would change the local label character to a colon (:). **L+** and **L-** are special formats that toggle the character between *adot* (+) and an @ sign (-). The default setting is **L-**.

**M - Enable/Disable Macro Instruction**

Set this option to **M-** and errors will be given on macro instructions, e.g. the following code:

```
li    r2,$587329
```

will expand to the following sequence:

```
lui   r2,$58
ori   r2,$7329
```

Set it to the default **M+** and the same code will generate an error.

**N - Generate an NOP in Branch Delay Slots**

The default setting for this option is **N-** but set it to **N+** and an NOP will be automatically inserted after all branch instructions.

**R - Allow \$ prefixed register names**

**R+** will allow the \$ prefixed versions of the register names to be used but will disable the use of \$ to prefix hexadecimal constants.

**S - Treat Equated Symbols as Labels**

The default setting for this option is **S-** but set it to **S+** and disassembly will treat equates as labels rather than just values.

**W - Print warning messages**

When this option is set to the default setting of **W+**, the Assembler will identify various instances where a warning message would be printed but assembly will continue. Disabling the **W** option will suppress the reporting of warning messages.

**WS - Allow white spaces**

The default setting for this option is **WS-** but if it is set to **WS+**, operands may contain white spaces. Thus, the statement:

```
dc.l    1 + 2
```

defines *alongwordof* value 1 with **WS** set **Off**, and a *longwordof* value 3 with **WS** set to **On**.

**X - XREFs in defined section**

The default setting for this option is **X-** but if it is set to **X+**, **XREFs** are assumed to be in the section in which they are defined. This allows optimisation to absolute word addressing to be performed provided that the section is defined with the **WORD** attribute or is in a **Group** with the **WORD** attribute.

## PUSHO and POPO

**Description** The **PUSHO** directive saves the current state of all the assembler options, and **POPO** restores the options to their previous state. They are used to make a temporary alteration to the state of one or more options.

**Syntax**                               **PUSHO**

**POPO**

**See Also**     **OPT**

**Examples**

```
pusho                                         ;save options state
opt                                         ws+, c+                     ;change options state

SetAlts         =                     height * time
SETALTS         dh                     256 * SetAlts

popo                                         ;restore previous state
```

---

## LIST and NOLIST

**Description** The **NOLIST** directive turns off listing generation; the **LIST** directive turns on the listing.

**Syntax**                               **NOLIST**

**LIST**                                 *indicator*

where indicator is a plus sign (+) or a minus sign (-).

**Remarks**

- If a list file is nominated, either by its inclusion on the Assembler command line, or in the Assembler's environment variable, a listing will be produced during the first pass.
- The Assembler maintains a *current listing status* variable, which is originally set to zero. List output is only generated when this variable is zero or positive. The listing directives affect the listing variable as follows:
  - **NOLIST** sets it to -1;
  - **LIST**, with no parameter, zeroes it;
  - **LIST +** adds 1;
  - **LIST -** subtracts 1.

<b>Examples</b>	<b>Directive</b>	<b>Status</b>	<b>Listing produced?</b>
	nolist	-1	no
	list -	-2	no
	list	0	yes
	list -	-1	no
	list -	-2	no
	list +	-1	no
	list +	0	yes

**Note:** The Assembler automatically suppresses production of listings during macro expansion and/or for unassembled code because of a failed conditional.

These actions can be overridden by:

- including the **/M** option on the Assembler command line to list expanding macros;
- including the **/C** option on the Assembler command line to list conditionally ignored code - see Command Line Syntax, chapter 2.



## INFORM and FAIL

**Description** The **INFORM** directive displays an error message contained in text which may optionally contain parameters to be substituted by the contents of expressions after evaluation. Further Assembler action is based upon the state of severity. **FAIL** directive is a pre-defined statement, included for compatibility with other Assemblers. It generates an "Assembly Failed" message and halts assembly.

**Syntax** **INFORM** *severity,text[,expression]*

**FAIL**

### Remarks

- These directives allow the programmer to display an appropriate message if an error condition is encountered which the Assembler does not recognise.
- Severity is in the range 0 to 3, with the following effects:
  - 0** : the Assembler simply displays the text;
  - 1** : the Assembler displays the text and issues a warning;
  - 2** : the Assembler displays the text and raises an error;
  - 3** : the Assembler displays the text, raises a fatal error and halts the assembly.
- *Text* may contain the parameters **%d**, **%h** and **%s**. They will be substituted by the **decimal**, **hex** or **string** values of the following expressions.

**Examples**

```

TableSize equ      TableEnd-TableStart
MaxTable  equ      512
if        TableSize>MaxTable
inform   0,"Table starts at %h and&
         is %h bytes long",&
         TableStart,TableSize
inform   3,"Table Limit Violation"
endif
  
```

## XDEF, XREF and PUBLIC

**Description** If several sub-programs are being linked, use **XDEF**, **XREF** and **PUBLIC** to refer to symbols in a sub-program which are defined in another sub-program.

**Syntax**

```

XDEF      symbol[,symbol]
XREF     symbol[,symbol]

PUBLIC    on
PUBLIC    off

```

### Remarks

- In the sub-program where symbols are initially defined, the **XDEF** directive is used to declare them as externals.
- In the sub-program which refers the symbols, the **XREF** directive is used to indicate that the symbols are in another sub-program.
- The Assembler does not completely evaluate an expression containing an **XREF**ed symbol; however, resolution will be effected by the linker.
- The **PUBLIC** directive allows the programmer to declare a number of symbols as externals. With a parameter of on, it tells the Assembler that all further symbols should be automatically **XDEF**ed, until a **PUBLIC** off is encountered.

**Examples** Sub-programA contains the following declarations :

```

xdef      Scores , Scorers
...

```

The corresponding declarations in sub-programB are:

```

xdef      PointsTable
xref      Scores , Scorers
...

```

---

```

Origin    public    on
          =          MainChar
Force     dh        speed*origin
Rebound   dh        45*angle
          public    off

```

## GLOBAL

**Description** The **GLOBAL** directive allows a symbol to be defined which will be treated as either an **XDEF** or an **XREF**. If a symbol is defined a**GLOBAL** and is later defined as a label, it will be treated as an**XDEF**. If the symbol is never defined, it will be treated as an **XREF**.

**Syntax**                                **GLOBAL**    *symbol*[,*symbol*]

**See Also**            **XREF, XDEF, PUBLIC**

**Remarks**            This is useful in header files because it allows all separately assembled sub-programs to share one header file, defining all global symbols. Any of these symbols later defined in a sub-program will be **XDEF**ed, the others will be treated as **XREF**s.



**CHAPTER 10****Debugger for Windows 95****Introduction**

The Debugger for Windows 95 takes advantage of the new range of 32-bit operating systems available for PCs; it provides full source level as well as traditional symbolic debugging and supports and enhances all the power of the DOS-based version plus the advantages of a multi-tasking GUI environment.

It helps you to detect, diagnose and correct errors in your programs via the step and trace facilities, with which you can examine local and global variables, registers and memory.

Breakpoints can be set wherever you need them at C and Assembler level and if required, these breaks can be made conditional on an expression. Additionally, selected breakpoints can be disabled for particular runs.

The Debugger employs drop-down menus, tool buttons, keyboard shortcuts and pop-up menus to help you debug quickly and intuitively.

**Projects**

The Debugger uses Projects to group together details of Files, Targets, Units, Views and other settings and preferences. All this information is saved and made available for your next debugging session.

**Views**

The Debugger offers the functionality of splitting the screen into a number of Panes, each displaying discreet or linked information. This information is available within a View, or document window (MDI Child). Each View can be split horizontally or vertically into the number of Panes you require and each Pane can be set to show a specific type of information.

You can have as many combinations of either tiled Panes or overlapping Views as you choose.

Your choice of Views depends on the level at which you are debugging. For example, it is appropriate to use a Register Pane for assembler debugging and a Local Pane when debugging in C.

Individual Views can be saved on disk for subsequent use in other Projects. However, when you close the Debugger and then re-start a session, your previous screen set-up will initially be displayed automatically.

### Colour Schemes

To aid identification, a **separate** colour scheme can be allocated to the Views used by each Unit that you reference. Alternatively, the same colour can be allocated **at** Views.

### Files

The Symbol Files you require are located and loaded by the Debugger and the relevant CPE and Binary Files are downloaded to the Target. Where a multi-unit system is in use you must also specify the Unit where Symbol and Binary Files are to be loaded.

### Dynamic Update

Changes in memory are highlighted on each display update, showing which areas of memory are being altered as the Target is being run and you are stepping and tracing your code.

The following topics are discussed in this chapter:

On-line Help
Installing the Debugger
Launching the Debugger
The File Server
Connecting the Target and Unit
Plug-In Components
Project Management
Debugger Productivity Features
Views
Panes
Debugging Options
Closing the Debugger

## On-line Help Available For The Debugger

Help text describing the features covered in this chapter, can also be accessed on-line via the Help menu on the main menu.

Selecting these options will result in the following:

- Contents will display the Contents page of the help system in the left-hand side of the screen. Clicking any of the underlined topics will provide further information about the relevant subject.
- Pane Types and the required Pane will directly access relevant text for the chosen Pane.
- Installation will display installation procedures.
- About will provide the Version Number.

Within the on-line help system, clicking text with **dotted** underline will display a pop-up description but double-clicking text with **solid** underline will display another (linked) help page.

The buttons at the top of the help text window can be used to facilitate the following:

- Search and/or Find to locate a particular word or topic.
- Back to re-display the previous page.
- $\leq\leq$  and  $\geq\geq$  to display the previous and next page in the browse sequence, as outlined in the Table Of Contents. (See below).
- Glossary to display an alphabetic listing of terms found in the help system. Click on any topic to obtain a pop-up definition.

As well as accessing information via the Contents page, on-line help can also be located via the Table Of Contents in the right-hand area of the screen. This represents the subject areas of the help system as book icons. Double-click any icon to display titles of the individual pages which compose each 'book'. Double-click any of these pages and the text will be displayed in the left-hand side of the screen.

## Installing The Debugger

A Set-up program is used to install the Debugger; this is distributed via either of the following methods:

- Full Release Files
- Maintenance Patch Files

Both methods are described in more detail below the Directory Structure.

### Directory Structure

All the Files relating to the Windows software live in one directory tree. This tree can reside anywhere but it is probably easier to locate it on the root of a local drive.

The default directory name is:

**'C:\PsyQ\_Win\'**

and it is *recommended* that you follow this convention. Set-up also installs several Files in the Windows System directory and adds two keys to the Registry.

These keys are:

[HKEY\_LOCAL\_MACHINE\SOFTWARE\SN Systems] (hardware settings)

[HKEY\_CURRENT\_USER\Software\SN Systems] (configuration information)

Set-up also registers the File types **psy** (Project), **.pqx** (plug-in) and **.cpe** and adds some programs to the Start menu.

**IMPORTANT:** Do not install the program on a server and execute it across a network. For un-installation advice, please contact SN Systems.



## Obtaining Releases And Patches

Releases and patches are available directly from SN Systems' BBS and ftp sites. In order to access these sites you will need an account with the necessary permissions.

To apply for an account telephone SN Systems or contact them via [Support@snsys.com](mailto:Support@snsys.com).

**Note:** Members of the Windows-Users mailing list will be notified of releases and patches as they become available.

### Determining The Latest Releases And Patches

This is achieved via any of the following methods:

- Contact [John@snsys.com](mailto:John@snsys.com)
- Look in one of the File sites for the latest Files and information
- See <http://www.snsys.com>

### Mailing Lists

SN Systems maintain a number of mailing lists for different purposes. For further information see <http://www.snsys.com>.

### Addresses for SN Systems' ftp, web and BBS sites

- <ftp://bbs.snsys.com>
- <http://www.snsys.com>
- BBS - +44 (0)117 9299 796 and +44 (0)117 9299 798

## Beta Test Scheme

SN Systems maintain a separate scheme for beta testing new versions of the Debugger.

The benefits of this are as follows:

- You will receive new versions of the Debugger before any other user
- You will have a prioritised chance to supply feedback to the Debugger's authors

If you are a member of this scheme, you don't need to install release versions of the Debugger.

For more information, contact [John@snsys.com](mailto:John@snsys.com).

## Installing A Full Release

A Full Release File contains an archive of several Files and a Set-up program that can be used to install the Debugger automatically.



To install the release:

1. Obtain the latest full release from SN Systems.
2. Read **Readme.txt** which contains last-minute installation instructions.
3. If the release is on a floppy, launch **Setup.exe** straight-away. If however, the release is in a zip File, you must unzip the File into a temporary directory and then launch Setup from that temporary directory.
4. If this is the first full installation of the Debugger, confirm the displayed license conditions.
5. Specify or confirm the directory in which you wish to install the Debugger.
6. The Files will be installed and the Registry will be updated.
7. Depending on the type of installation, specify the settings for the DEX Board or SCSI Card. (See **Configuring Your Dex Board/SCSI Card** below).
8. Once the dialogue has been completed the installation is complete.

**Note:** This method can be used for the first installation of the Debugger and also for subsequent upgrades if you do not wish to use Maintenance Patches. See **Upgrading Your System** below.

## Upgrading Your System

From time to time, SN Systems will provide updates to the Debugger that introduce bug fixes and new features. For your convenience, updates are supplied as full installations **and** as maintenance patches.

A Maintenance Patch contains only the difference between Files so it is much smaller. This makes it quicker to download and apply. However, patches can only be applied over certain previous versions.



To apply a Maintenance Patch:

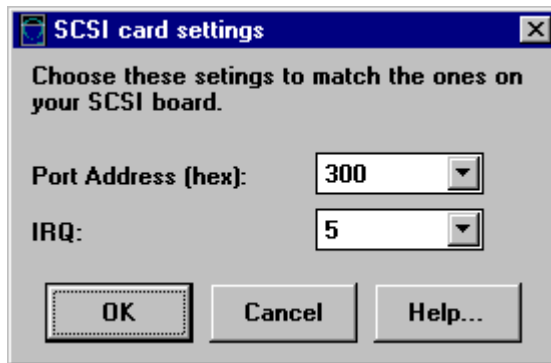
1. Determine your current release by reading the **About** box for the Debugger.
2. Obtain the Maintenance Patch from SN Systems. Instructions will be provided so you can determine which patch must be applied.
3. Apply the patch according to the instructions.

## Configuring Your SCSI Card


If you are installing a Full Release for a SCSI Target, you must specify the settings for the SCSI Card.



Enter appropriate values to the dialogue box displayed during the Set-up program



*SCSI Card Settings Dialogue Box*

1. Specify a **Port Address** and **IRQ** value by clicking on the down arrows and selecting as appropriate.
2. Click .

The installation is now complete.

**IMPORTANT:** **Port Address** and **IRQ** values must be correct for the Debugger to work. If they are incorrect or another device is configured to use similar settings, the programs will not work.

**Note:** The **IRQ** value can be set to 0 to run without interrupts. This will help with trouble shooting and will only impair the performance of the system if you make a lot of use of file or message serving.

## Testing The Installation

► Once the Debugger has been installed, you should test the installation as follows:

1. Ensure that the Nintendo 64 is switched off and that the Target Adapter and game cartridge are connected to it. The power should be on for the Adapter and the bottom LED should be flashing **slowly**, indicating that the bios has not yet been loaded..
2. Run the File Server by selecting the Debugger File Server from the Start menu.
3. This will attempt to connect to the Nintendo 64 console. If the connection is successful the File Server will download the bios; the bottom LED will go out and the following text will be output from the File Server:

```
Psy-Q File and Message Server, Copyright 1996, 1997, SN Systems Ltd  
Version 2.0 (January 1997)  
Release 10, Patch Level 3
```

```
Target Found: Bus ID = 0, SCSI ID = 0  
New Downloader - Reading profile information...  
Profile read for Nintendo 64 (no bios)  
OK  
Rebooting the Nintendo 64 (no bios)...  
Getting the target's ID...  
ID is N64-NO-BIOS PLEASE LOAD  
bios1...  
sleep .5 sec...  
bios2...  
sleep .5 sec...  
New Downloader - Reading profile information...  
Profile read for Nintendo 64
```

If this has occurred communication will have been established between the PC and the Nintendo 64.

4. Now you are ready to download a test cartridge image into the cartridge RAM.

Proceed as follows:

5. From the File Server select the Tools menu.
6. Select the Upload option.
7. Click the Download radio button and change the **Download Address** to 0xB0000000 .

8. Click **B**rowse and select the file `test.bin` from the `psyq-win\examples\n64` directory.
9. Click OK.

The downloaded image is now in the cartridge RAM.

10. Switch on the Nintendo 64. The system will boot the image and stop at a breakpoint at the start of the program.
11. Run the Debugger from the Start menu.
12. You are now ready to begin debugging.

## Documentation

If you experience problems during installation, the following documents provide useful information:

- **README.HTM**
- **README.RTF**
- **README.N64**



## Using The Check System Diagnostic Tool

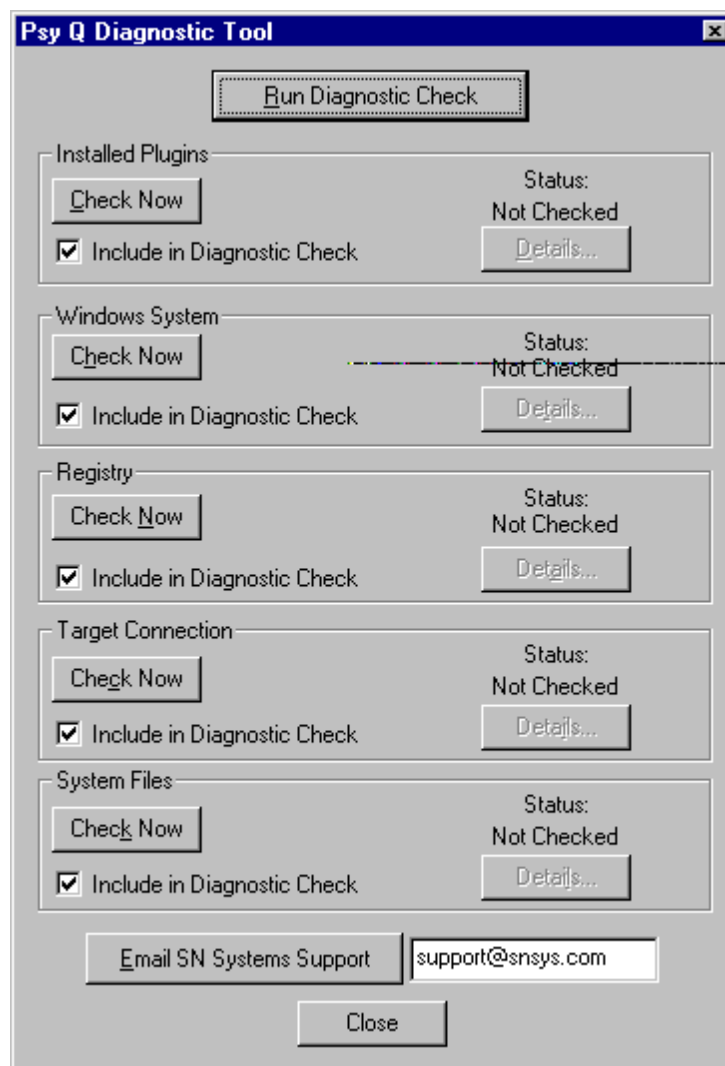
The Check System Diagnostic Tool is a Plug-In which is used to check for faults in the Debugger installation or hardware and if any are found and a compatible Email program is in use, to automatically generate a report of them to SN Systems Support.



There are two ways to access the Diagnostic Tool:

1. Either, first select the Debugger group from the Program menu and then the **Check System for Errors** option.

Alternatively, select **System Diagnostic Tool** from the main **Tools** menu.



Via the displayed Diagnostic Tool Dialog you can check the following areas:

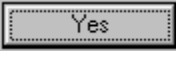
<b>Plug-Ins -</b>	That the required Plug-Ins are installed and that files are actually present for registered Plug-Ins.
<b>Windows System -</b>	That the required DLLs are present and are the correct size and version.
<b>Registry -</b>	That the registry settings are correct.
<b>Target Connection -</b>	That the Target is communicating with the PC.
<b>System Files -</b>	That a list of the installed Psy-Win files has been printed.

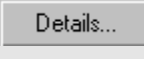
The results of any previous diagnostic checks will be shown for each group.

- By default, the diagnostic process will include all the above groups. To exclude a particular check or checks from the run, de-select **Include in Diagnostic Check** for each relevant group.


- Click the  button.


If any errors are detected you will be prompted to mail SN Systems support.

- Provided that you have a MAPI compatible Email program and it is currently running, click  and your email window will appear. Each attached file to be sent to SN Systems support contains error information for one of the checks.

**Note:** Click the required  button on the Diagnostic Tool Dialog to examine these details yourself.

- Enter a description of the error(s) and send the mail as normal.

- Click the appropriate  button to repeat any particular check.

- If no errors are detected but you still suspect problems, click the  button to re-run the diagnostic checks and manually access the email program. Send a message as required.

## Launching The Debugger

There are several ways of launching the Debugger under Windows 95.



A simple way is as follows

1. Select the **Start** menu from Windows 95.
2. Choose the **P**rograms option from the list displayed.
3. Select the **Psy-Q** folder from the list of programs.
4. Select **D**ebugger from the folder.

You can also launch the Debugger from the desktop or folders or through Explorer in Windows 95.

When you close the Debugger, it will remember the current working directory and restore this when it is re-opened.

The Debugger will also remember the current state of the flashing caret when it is loaded. If you turn the flashing off, it will remain off until you close the Debugger, when it will be restored. If you then re-open the Debugger, the caret flashing will be stopped again.

**Note:** Switching away from the Debugger will not re-start the caret flashing.

With the drag and drop facility you can drop a Debugger Project File (extension .PSY) onto the icon of the Debugger and the selected Project is launched.

Alternatively, as file type .PSY has been registered with the Windows 95 shell, you can right-click on a Project File, select **D**ebug from the menu and the Debugger will be launched with the selected Project.

**Note:** While the Debugger is still running, you can open **n**ew Project by following the procedure described in the previous paragraph.

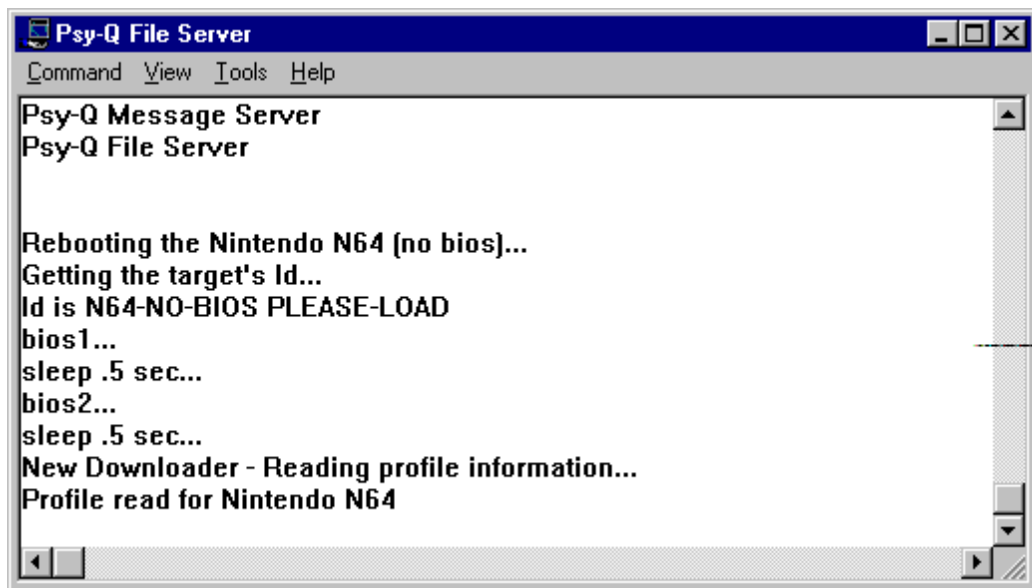
When you launch the Debugger it scans for recognised Units and if none are found a dialogue box prompts you to either **R**epoll or **Q**uit. If **R**epoll does not work you are advised to start troubleshooting.

## The File Server

The primary function of the File Server is to provide the PC Open and PC Read functions for your program.

When the File Server is running, the icon and name of the application appear on the Task bar of Windows 95.

You can view the messages appended into the message window of the File Server during debugging by clicking on this icon.



*File Server Message Window*

If you wish the message window to be permanently displayed on top of other windows, select **Always on Top** from the **View** menu.

When the Debugger or File server experiences a communication error, the screens will go blank. Press **Ctrl + U** to re-attempt the connection.

## File Server Menu Commands

In addition, the following options are available from the File Server Command menu:

- **Run Project** - Not currently in use
- **Debug Project** - Not currently in use..
- **Download CPE File** to the Target.
- **Run CPE** runs the Target after the CPE File has been downloaded.
- **Ping** determines the current status of the Target
- **Halt** provides the option to stop the Target if it is running.
- **Start** causes the Target to start running.
- **Clear Window** removes any File Server messages.
- **Reboot Target** reboots the Target.
- **Reset Target** will attempt a hardware reset if supported by the Target.
- **File Serving** determines whether file and message serving are available.
- **Telnet Server** determines whether the Telnet Server is accessible.

**Note:** Resetting the Target while the Debugger is running may cause unpredictable results.

**Note:** The Reset option is also available from the System menu of the File server.

**Note:** When you close the File Server, it will remember the current working directory and restore this when it is re-opened.

## Connecting The Target and Unit

The Debugger automatically checks your system when you launch it, identifies any Targets that are connected and according to whether you are running a single or multi-Unit system, automatically connects to the relevant Unit(s).

The Unit toolbar appears at the far right of the Main Menu bar. The last icon in the toolbar has a pictogram of the Target known as the Unit button.

There will be a Unit toolbar and unique button for each Unit identified. Click on the button to display the Unit menu. This menu allows you to download and load (as relevant) foreign CPE and foreign Symbol Files and download non-foreign CPE and Binary Files.

The Unit button menu options are:

- Download CPE
- Download Binary
- Load Symbols

Each toolbar contains a set of debugging icons which represent:

- Starting programs
- Stopping a program running
- Stepping into a subroutine
- Stepping over a subroutine
- Stepping out of a subroutine

**Note:** Note that these actions operate only in respect of the relevant Unit; therefore, where a multi-Unit system is in use they will not necessarily operate in respect of the Active View.

**Note:** The File Server window displays any output from the Target while it is running.

## Plug-In Components

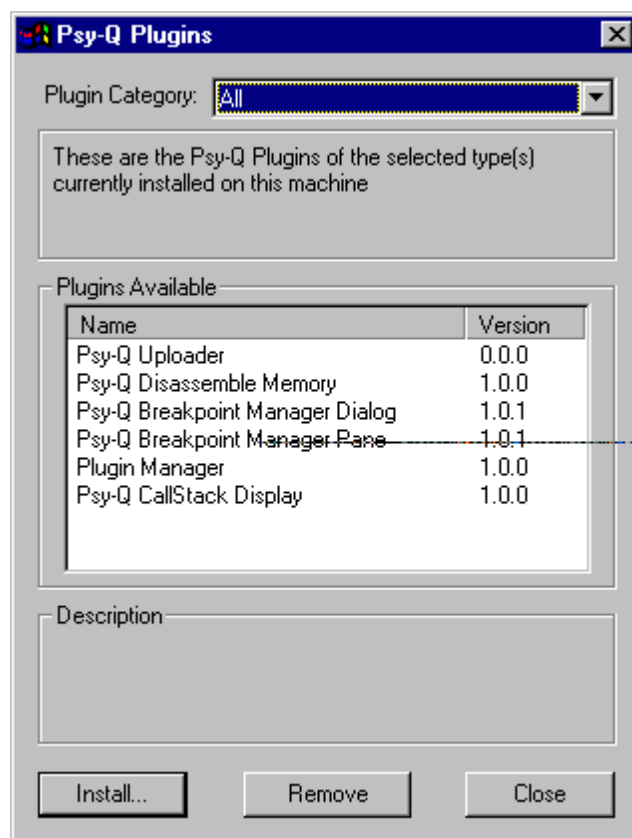
The Debugger provides two types of Plug-In components; these provide additional functionality or extend the Debugger in some way and are installed separately. They are categorised as follows:

- Tool Based Plug-Ins
- Pane Based Plug-Ins





To install or remove a Plug-In component:

1. Select the Plug-In Manager from the **Tools** menu.



*Plug-In Manager Installation/Removal Mode*

2. This dialog allows you to install or remove all the Plug-Ins currently available on your machine. If you wish to restrict the display to Tool-based or Pane-based versions only, select the relevant group from the **Plug-In Category** pull-down menu.
3. Select as required and click  or  as necessary.

You will now be able to access the installed Plug-Ins.

### Tool Based Plug-Ins

The following facilities are available:

- Plug-In Manager
- Check System Diagnostics
- Upload/Download Memory
- Disassemble Memory
- View and edit Breakpoints



To access a Tools-based Plug-In component:

From the main Tools menu, select the required option.

A relevant dialog box will be provided for the entry and inspection of information.

### Pane Based Plug-Ins

The following facilities are available:

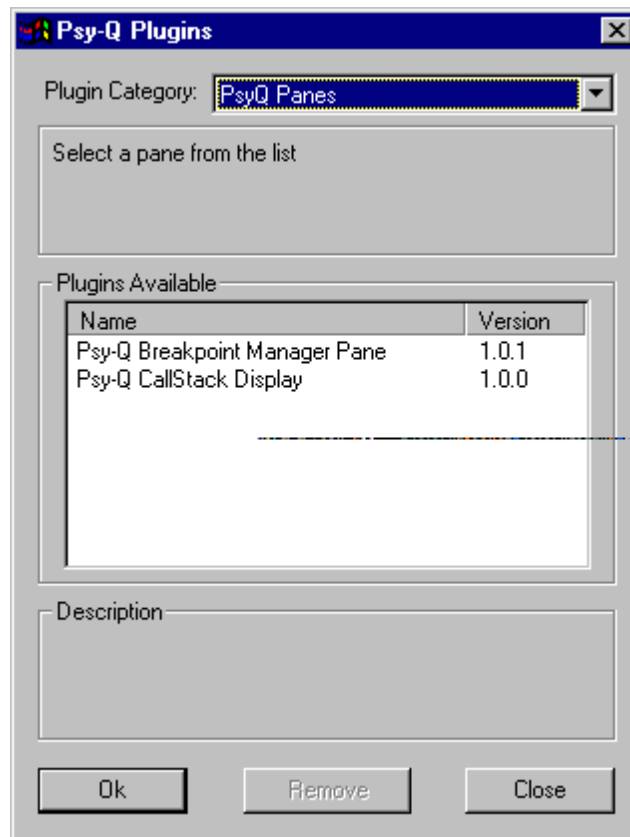
- Plug-In Manager
- Breakpoint Manager
- Call-Stack Display



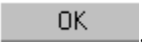
To access a Pane based Plug-In:

1. Right-click in a relevant Pane.
2. Select Change Pane.
3. Select Other or **Ctrl + Shift + O**.
4. The Plug-In Manager will display the currently available Pane Plug-Ins.





*Plug-In Manager When Switching Between Panes*

5. Select as required and click .

Information from the selected option will be displayed in the Pane.

**Note:** The functionality of the Breakpoint Manager Pane is the same as for the Tool based Breakpoint Dialog.

6. To switch to another Pane based Plug-In, repeat step **1 to 5**.

## Using The Disassemble Memory Dialog

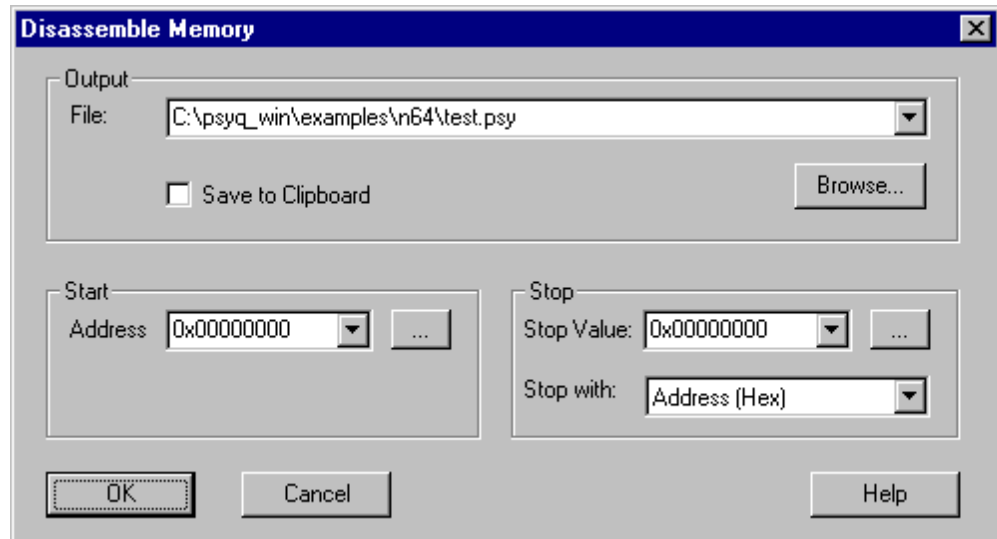
The Disassemble Memory Dialog is a Tools based Plug-In which is used to produce a machine code disassembly of a specified area of memory, similar to that which appears in the Disassembly Pane.

The area to be disassembled can be specified as an address range, as a number of bytes or as a number of instructions. The output can be copied to a file on your PC or it can be sent to the Windows clipboard to be pasted directly into another Windows program.

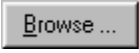


To access the Disassemble Memory Dialog:


1. From the main Tools menu select the Disassemble Memory option.




2. Specify where the disassembled memory is to be output.


Where it is to be output to a file the name and path can be explicitly entered or click  and select as required.

Alternatively, to save to the Windows clipboard check the **Save to Clipboard** box.

3. Specify the Start Address, i.e. the address on the Target's memory from where to start the disassembly. This value is always in hexadecimal. Click on the adjacent  button to call up the Expression Manager; from here you can specify an expression to be used as the start address.

4. Specify a **Stop Value**. If the **Stop With** entry is an address, the Stop Value must be in **hexadecimal**. If the entry to Stop With is a number of bytes or instructions however, the Stop Value must be **decimal**.

When the Stop Value is specified as **address**, the Expression Manager is available via the  button.

5. Click  when the values have been specified.

## Using The Upload/Download Memory Tool

The Upload/Download Memory Dialog is a Tools based Plug-In which allows you to:

- Upload a portion of memory from the target machine to a file on the host machine

or

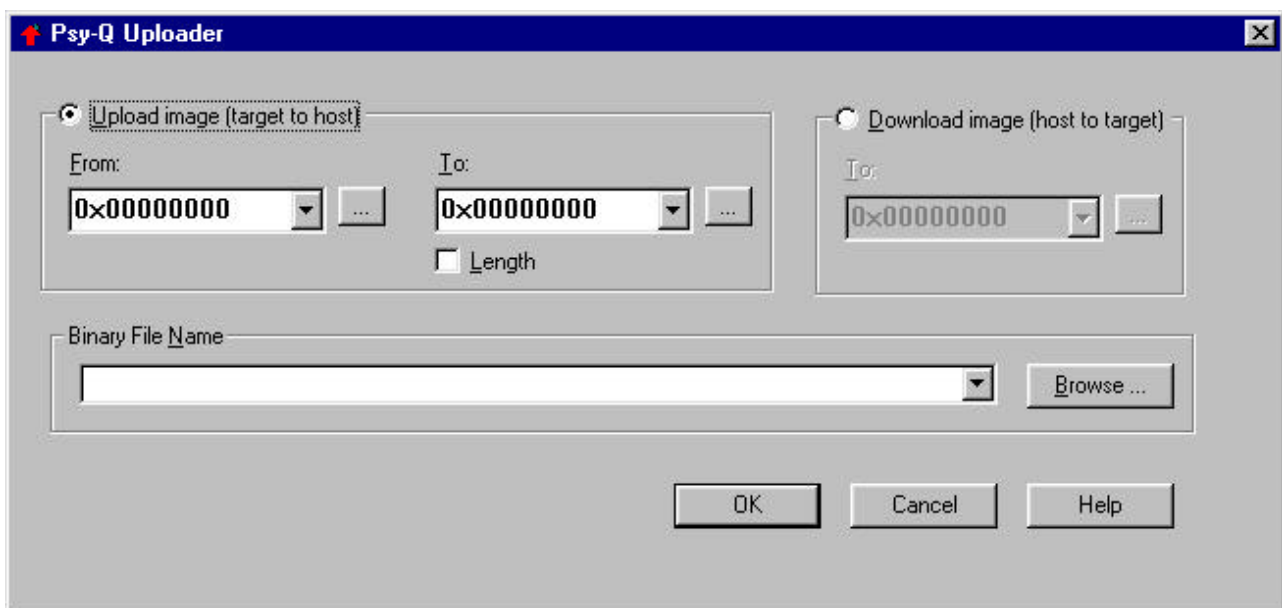
- Download a portion of memory from the host machine to the target machine

It is then possible to save and restore the state that a machine's memory was in prior to an exception occurring.



To access the Uploader/Download Memory Dialog:

1. From the main **T**ools menu, select the Upload/Download Memory option.
2. Select the required mode from the relevant **U**pload/**D**ownload **M**emory radio buttons.





*Upload/Download Dialog With Upload Image Selected.*

Only one option can be selected at a time and the default mode is **U**pload.


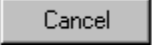
When **U**pload Image is selected you must specify the **F**rom and **T**o positions of the required image or check the **L**ength box and **o**nly specify a **F**rom position if you wish this to represent the length of a block of memory.

When **Download Image** is selected you only need to specify the **file address** as the memory to be downloaded is calculated from the size of the file from which memory is being downloaded.

**Note:** Clicking the  button will invoke the Expression Manager via which you can search for and specify particular addresses.

3. Enter a name to the Binary File box; this can be the complete name and path or click  and select as appropriate.
4. If you are uploading memory you will be asked to supply a filename to which the portion of memory should be saved. If you are downloading memory, you must specify the name of the file from which the memory will be taken.

**Note:** If you click on the downward arrow adjacent to each edit box, a list of previously used addresses or files (as relevant) will be displayed. The most recently used items will appear at the top of the list.

5. Click  and an attempt is made to upload or download the memory. Errors resulting from this action will be displayed in the File Server window. Click  to exit the dialog without uploading or downloading memory.

## Projects

A Project is a combination of the elements and settings associated with a specific development project.

It consists of any or all of the following:

- Units to be debugged
- Screen layout
- CPE Files
- Symbol Files
- Binary Files
- Breakpoints
- Other settings and preferences.

This set of information is used by the Debugger to track the debugging process. When you save a Project this includes all the Views, colour schemes and breakpoints already specified for it. These settings are reinstated when the Project is next opened.

### Setting Up And Managing Projects



To create a new Project you can either:

1. Open the default Project by selecting **N**ew from the **P**roject menu.
  2. Save and name the Project.
- or
1. As 1) above.
  2. Select files for the Project and add them to the file list.
  3. Set file properties for executable files.
  4. Save and name the Project
  5. Re-open the Project with the files in the file list.

## Selecting Files For Your Project

The Debugger uses files that are output from the build process. Three types of file may be included in the Project; these are:

- CPE Executable Files
- Symbol Files
- Binary Files

### Adding Files To The List Of Project Files



This is achieved as follows

1. Select the Project menu from the Menu bar.
2. Choose Files from the menu; the Files dialog appears.
3. Click  to insert them into your file list.
4. Select CPE, Binary or Symbol Files from the **Files of Type**' drop-down list.
5. Locate the file and click .
6. When you add a file to the file list a relevant dialog box requests you to set the file properties. For CPE and Binary Files these will determine the downloading of files to the Target. Additionally, for Binary and Symbol Files, they determine the Unit to which they will be loaded. See the **Understanding File Properties**' sections below.

**Note:** It is not necessary to specify the Unit to which a CPE File should be loaded as this information is held within the file itself.

7. Repeat the operation until all the files you require appear in the list. To remove a file from the list, highlight it and click .
8. Click  when you have added all the files you require.

The CPE and Binary Files will be downloaded in the order shown in the file list.

**Note:** As file type .CPE has been registered with the Windows 95 shell, you can run a program directly from the shell by double-clicking on the relevant CPE File. Alternatively, if you wish to download the file to the Target without running it, right-click the relevant file and select **D**ownload from the menu.

**Note:** When you add Binary and Symbol Files to a Project they are not loaded until the Project is saved and re-opened.

## Changing The Order Of Files In The File List

If you have multiple CPE and Binary Files within your Project, the order in which they are loaded during debugging is determined by the position you placed them in the File list.



To change the file sequence

1. Select the **P**roject menu from the Menu bar.
2. Choose **F**iles from the menu.
3. Highlight a file.
4. Use **P**romote or **D**emote to alter the position of the file in the list.

Repeat the process until the files are in the required order.

**Note:** This option is only useful if you have multiple CPE and Binary Files in your Project and the load order is important.



## Specifying CPE File Properties

When you select a CPE File to include in your Project, a dialogue box requests that you set the properties for this file.

These properties allow you to control the downloading of files to the Target. The options are:



- **Download when Project starts-** This causes the CPE File to be downloaded when the Project is opened or reopened.
- **Run after CPE has been downloaded-** This causes the Unit to start running the code after downloading the file.

You may select either or both of these properties for any CPE File in the Project.

If you do not set the properties of at least one CPE File, the Debugger will not download any files to the Target when the Project is opened.



To change CPE File properties:



1. Select the Project menu from the Menu bar.
2. Choose Files from the menu.
3. Select the CPE File to change.
4. Click .
5. Use the check boxes to apply the properties.
6. Click .

## Specifying Symbol File Properties

When you select a Symbol File to include in your Project, a dialogue box requests that you confirm or specify the Unit to which the file should be loaded.

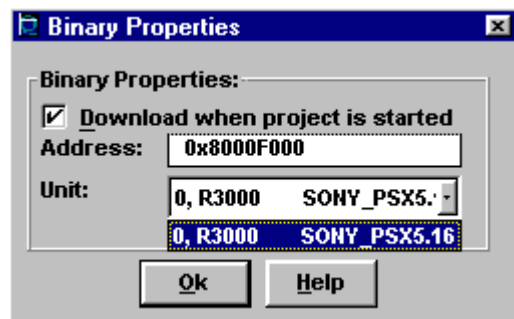


To change Symbol File properties

1. If the required Unit is not already displayed, click the down arrow until it appears.
2. Highlight the required Unit.
3. Click .
4. Click .

## Specifying Binary File Properties

When you select a Binary File to include in your Project, you must complete the following dialogue box:



*Binary File Properties Dialogue Box*

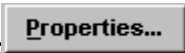

These properties allow you to control the downloading of files to the Target:

- **Download when Project starts-** If this is selected the Binary File will be downloaded when the Project is opened or reopened.
- **Downloaded to a specified address-** The files will be downloaded to the address specified. This should be in OX notation for hexadecimal numbers. The default address will be zero.
- **Specify the Unit where the File is to be loaded-** Click on the down arrow to display further Units.

If you do not set the first option for at least one Binary File, the File Server will not download any Binary Files to the Target when the Project is opened. However, all Binary Files in the Project will be available on the relevant Unit menu.



To change Binary File properties

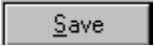
1. Select the Project menu from the Menu bar.
2. Choose Files from the menu.
3. Select the Binary File to change and click .
4. Select the **Download when Project starts** option if required and/or enter a relevant **address**.
5. Confirm or specify the Unit where the File is to be loaded.
6. Click .

## Saving Your Project

Once the files have been selected, the new Project must be saved and re-loaded before debugging can begin.



This is achieved as follows

1. Select the Project menu from the Menu bar.
2. Choose the Save option from the menu.
3. Give a name and path to your Project.
4. File names in Windows 95 are up to 250 characters long and can contain spaces. Debugger Project Files must be saved with the default File extension **.PSY**.
5. Click .

**Note:** For a new Project you can choose the Restore rather than the Save option. Restore prompts you to save the Project before reloading it.

**Note:** The Save or Save As options can be used to save an existing Project.


## Re-opening A Project

After saving a new Project you must re-open it before working with the files which have been added to the file list.



This is achieved as follows

1. Select the Project menu from the Menu bar.
2. Choose the Re-open option from the menu.

**Note:** **Ctrl + R** or the Re-open icon on the toolbar  can also be used to re-open a Project.

**Note:** When you close the Debugger it will remember the current working directory and restore this when it is opened.

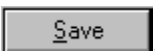
## Saving A Project Under A New Name

The **Save As** option on the Project menu is used to save changes made to an existing Project, under a new name.

The default File extension for a Debugger Project is **PSY**. When you save Project Files you must use this extension.



To save a Project under a new name

1. Select the **P**roject menu from the Menu bar.
2. Choose the **S**ave option from the menu.
3. Give a name and path to the renamed Project.
4. Click .

## Restoring A Project

The **R**estore option on the Project menu is used to re-load a Project in the state in which it was last saved, abandoning any changes made since the last save.



To restore a Project

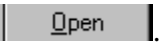
1. Select the **P**roject menu from the Menu bar.
2. Choose the **R**estore option from the menu.

## Opening An Existing Project

When you launch the Debugger, the last Project you worked on will be loaded automatically.




To open a **different** Project:

1. Select the Project menu from the Menu bar.
2. Choose the Open option from the menu.
3. Select the Project (.PSY) you require.
4. Click .

Or

From the Project menu select the required Project from the Project File history listing..

**Note:** An existing Project can also be opened via the Open Project icon  found on the toolbar.


**Note:** As File type .PSY has been registered with the Windows 95 shell, you can run a Project by double-clicking on the relevant .PSY File within the shell. Alternatively, if you only wish to load the Project into the Debugger, right-click the relevant file and select Debug from the menu.

## Manually Loading Files Into A Project

External Files can be downloaded at any time; they are not saved with the Project.




External **CPE** Files are **downloaded** to the Target as follows:

1. Click on the Unit menu at the base of the Debugger screen.
2. Choose the Download CPE option from the menu.
3. Choose the External File option.
4. Browse and select the required CPE File.
5. Click .

**Note:** You can also download a CPE File by double/clicking it within the shell.



**Symbol** Files can **be loaded** into the Debugger as follows:

1. Click on the relevant Unit menu at the base of the Debugger window.
2. Choose the Load Symbols option from the menu.
3. Browse and select the required Symbol File.
4. Click .

**Note:** An error message will be displayed if you attempt to load Symbol Files with overlays. This facility will be supported in a later version of the software.

## The Debugger Productivity Features

To enable you to work faster and more efficiently when using the Debugger, the following two features speed up your control of the debugging runs.

- Toolbar Icons
- Hot Keys

### Toolbar Icons



The toolbar contains the group of icons shown above. Icons provide a quicker means of activating commands and setting properties.

From left to right they represent the following actions:

Open a Project File  
Save and then reopen the current Project  
Open a new View  
Switch to the next View  
Split the Active Pane horizontally  
Split the Active Pane vertically  
Delete the Active Pane  
Set the default colour scheme

The Show **T**oolbar option on the **P**roject menu is used to toggle the menu bar on and off. When the option is ticked the toolbar is displayed.



To toggle the toolbar

1. Select the **P**roject menu from the Menu bar.
2. Choose the Show **T**oolbar / Hide **T**oolbar option from the menu.

**Note:** Every Pane type has its own, additional toolbar which is appended to the main toolbar when that Pane is made Active.

**Note:** Double-click to customise a toolbar or use **Alt+Drag** to move buttons around the toolbar or delete them.



## Hot Keys

The following Hot Keys can be used instead of the Debugger menu options:

<b>F2</b>	Split Horizontal.
<b>F3</b>	Split Vertical.
<b>F4</b>	Delete current Pane.
<b>F5</b>	Toggle breakpoint on and off.
<b>F6</b>	Run to cursor.
<b>F7</b>	Step into a subroutine.
<b>F8</b>	Step over a subroutine.
<b>F9</b>	Run a program.
<b>F12</b>	Step out of a subroutine.
<b>Esc</b>	Stop a program running.
<b>Ctrl + A</b>	Add a watch to a Watch Pane.
<b>Ctrl + B</b>	Add horizontal scroll bars to Expression, Disassembly and Source Panes.
<b>Ctrl + L</b>	Add locals to a Watch Pane. This command adds all locals to the watch but when the watch goes out of scope it will not delete them as in a Local Pane.
<b>Ctrl + M</b>	Toggle the stepping mode between Source and Disassembly
<b>Ctrl + R</b>	To Re-open.
<b>Ctrl + Shift + D</b>	Change Pane to Disassembly Pane.
<b>Ctrl + Shift + L</b>	Change Pane to Local Pane.
<b>Ctrl + Shift + M</b>	Change Pane to Memory Pane.
<b>Ctrl + Shift + R</b>	Change Pane to Register Pane.
<b>Ctrl + Shift + S</b>	Change Pane to Source Pane.
<b>Ctrl + Shift + T</b>	Toggle between hexadecimal and decimal globally.
<b>Ctrl + T</b>	Change the mode of a single expression from hex to decimal or vice versa.
<b>Ctrl + Shift + W</b>	Change Pane to Watch Pane.
<b>Ctrl + Shift + number</b>	Assign an identifying number to each View.
<b>Ctrl + number</b>	Switch between Views.
<b>Shift + Arrow Keys</b>	Activate adjacent Pane in the specified direction. Where more than one, the current caret position determines the Pane to be made Active.
<b>Shift + Ctrl + F10</b>	Set the width to match the size of the window in the Memory Pane.
<b>Ins</b>	New View or add a watch in a Watch Pane or add a breakpoint in the Breakpoint Manager.
<b>Alt + Shift + number</b>	Assign an identifying number to each Plug-In Pane.
<b>Alt + number</b>	Switch between Plug-In Panes.

**Note:** These keys will all operate in respect of the **Active** Pane.

## Views

A View appears in the main window of the Debugger; it is used to display debugging information according to your requirements and to control step and trace actions during debugging.

When a Project is first created it has a default Pane layout.

Views can be split into as many Panes as you wish. These can be of the same or different types.

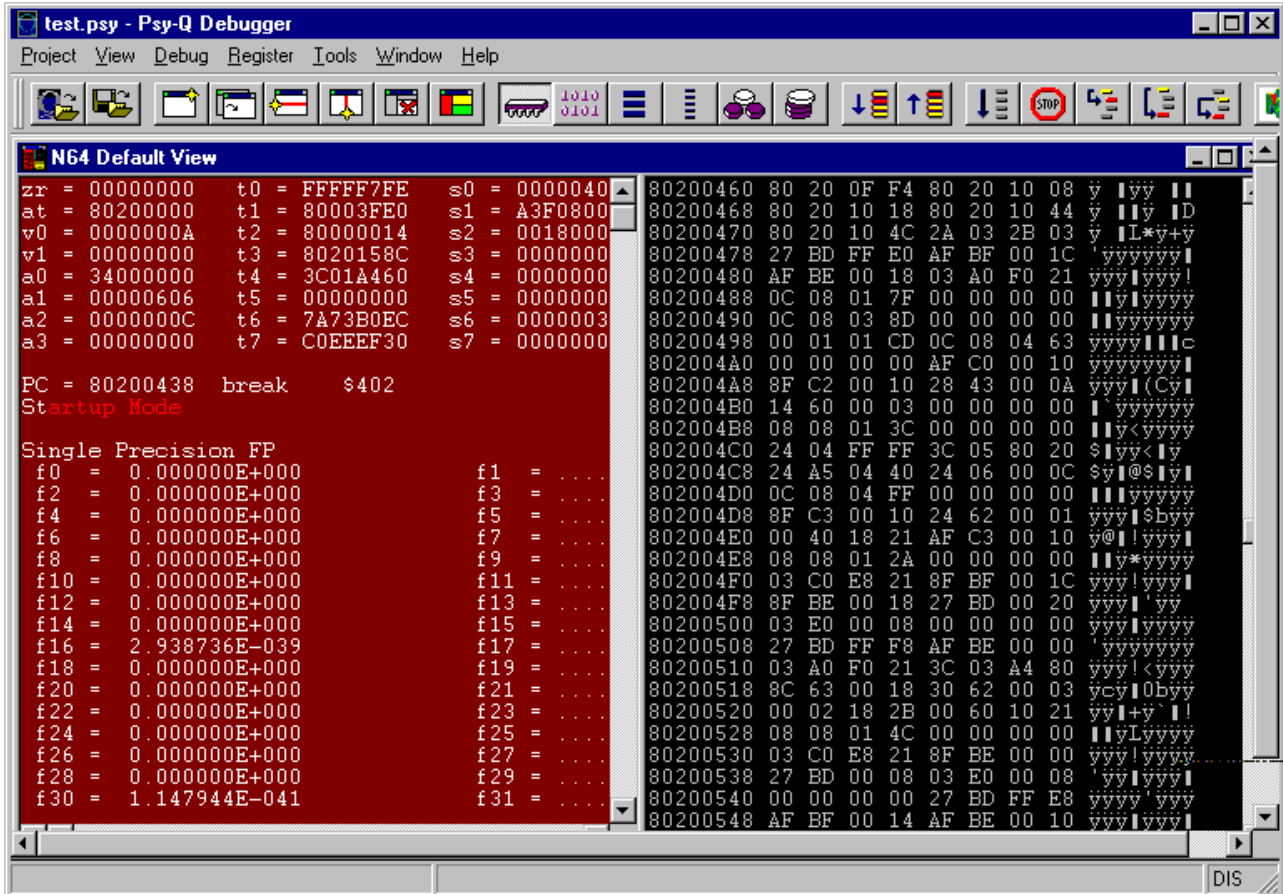
Only one is Active at any time; it will be displayed in a different colour scheme to the others.

**Notes:** Having created a View of different Panes you can save this as a View File either in, or independent of, the Project. Further information about Panes can be found in **Working With Panes** and **Selecting A Pane Type**

## Creating A View

Within a Project you can create as many Views as required; in turn, each View can be split into as many Panes as you need.

When you open a new Project, one View is displayed for each Unit connected.




Default View



To create a new View:

1. Select the View menu from the Menu bar.
2. Choose the New option from the menu.
3. From the Choose Unit box specify the Unit for which you wish to create a new View.

**Note:** The Choose Unit box will not appear when you are connected to **single** Unit.

You can also use the New View icon on the toolbar  to create a new View or use the Hot Key **Insert**.

Alternatively, you can open a new View from the relevant Unit button, in which case you won't be prompted for the required Unit.

**Note:** A new View is supplied with the title 'Default View'. The **View Name** option in the View menu should be used to give it a title.

**Note:** Views can be saved either inside or outside of Projects.


## Cycling between Views



If you have more than one View open within a Project you can cycle between them as follows:

1. Select the **View** menu from the Menu bar.
2. Choose the **Next View** option.

The Views are cycled around until you see the one you require. All Views appear on the View list regardless of the Unit for which they have been specified.

Alternatively, the Next View icon on the toolbar , the Hot Keys **Ctrl + F6** or **Ctrl + TAB** can be used to cycle between Views.

Up to ten Views can each be assigned an identifying number from 0 to 9; the number of the assignment will appear in the title bar of the View. You can then use the number to quickly switch between Views. Select **Ctrl + Shift + number** to assign the number and **Ctrl + number** to access the View. If the destination View is minimised, it will be restored or maximised according to the state of the current View.

**Note:** These assignments are preserved in Projects and in saved Views.

## Saving Your Views

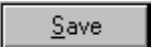
Any number of Views can be saved within a Project.

All open Views will automatically be saved when you save the Project and will be opened when the Project is re-opened.

View Files can also be saved independently of Projects using the **Save As** command on the View menu.



This is achieved as follows:


1. Arrange the Panes as you require.
2. Select the **V**iew menu from the Menu bar.
3. Choose the **S**ave **A**s option from the menu.
4. Give the View a name and path.
5. Click .

**Note:** The name you give the View File is not displayed on the View. To give a View a title use the **V**iew Name option on the **V**iew menu.

## Naming A View



Because you can use many Views within a Project, it is helpful to give each View an individual title.

1. Select the View menu from the Menu bar.
2. Choose the View Name option from the menu.
3. Enter the View name in the edit box.
4. Click . The name appears at the top of the View.

**Note:** This is not the name of the File. See the note **Saving Your Views** above for further details.

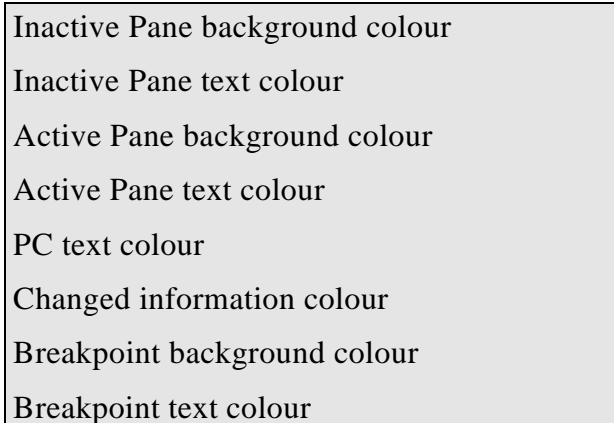
## Changing Colour Schemes In Views



To change the colours for a particular Unit

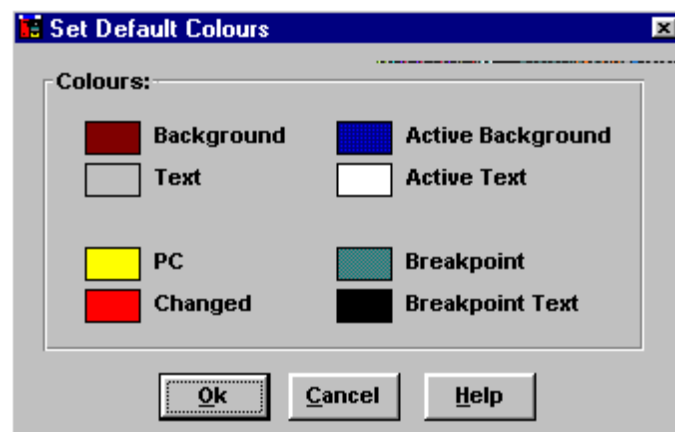
1. Activate a View/Pane on the Unit that you wish to set colours for.
2. Select the View menu from the Menu bar.
3. Choose Set Default Colours... from the menu.

The following areas may be changed for the Active Unit:



4. Click on the box representing the area you wish to amend.

A standard Windows dialogue box allows you to choose from a range of standard or customised colours.




*Set Default Colours Dialogue Box*

5. Select the required colour(s).

The selected colour scheme will be displayed for all visible Views.

6. Select  to retain the revised colours or  to revert to the original scheme.

Unit colours can also be amended by clicking on the Set Colour icon  on the toolbar.

## Working With Panes

When a Project is first set up, the default View contains a default Pane layout for each Unit connected. However, this View can be split into as many Panes as you wish. These can be of the same or different types. Only one of the Panes is Active; it will be displayed in a different colour scheme to the others.



A Pane can be made Active via any of the following methods:

- Clicking on it
- Changing the Active View and the first Pane created for that View will become Active
- Using **Shift** and the appropriate **arrow key** to Activate the Pane in the specified direction
- Clicking the right mouse button on the required Pane and selecting from the displayed menu

## Splitting Panes



A View can be divided into as many Panes as you wish. Click on the one you wish to split to make it Active, then

1. Select the View menu from the Menu bar.
2. Choose either SplitVertical or SplitHorizontal from the menu.

The Active Pane is split in half, either vertically or horizontally, depending on your choice.

You can also split a Pane horizontally or vertically via the icons on the toolbar



or by using the hot keys **F2** to split horizontally or **F3** to split vertically.

**Note:** When you split a Pane the two halves will both be of the same type as the original. The font for the new Pane will also match that of the original.



## Changing Pane Sizes



To change the size of Panes:

Drag the splitter bar between the Panes with the mouse.

The size and position of the Panes is saved when you save the View or the Project.


**Note:** Splitter bars only control the areas between the Panes. If you wish to change the size of the Debugger window you have to use the borders of the window itself.

## Deleting A Pane



The Delete Pane option on the View menu is used to delete a Pane within a View, as follows:

1. Click on the required Pane to make it Active.
2. Select the View menu from the Menu bar.
3. Choose Delete Pane from the menu.

Alternatively, the Delete Pane icon on the toolbar  or the hot key **F4** can be used to delete the Active Pane.

## Changing Fonts In Panes



If required, the SetFont command can be used to change the display of text within a Pane, as follows

1. Make the required Pane Active.
2. Select the View menu from the menu bar.
3. Choose SetFont from the menu.

A standard Windows dialogue box allows you to select from the available fonts.

**Note:** When you split a Pane, the new Pane will be displayed in the same font as the original one.

**IMPORTANT:** You will only be able to use non proportional fonts, e.g. Courier, New Courier, Fixed Sys, Terminal..

**See Also:**

**Changing Colour Schemes In Views**

## Scrolling Within A Pane

Many Panes are unable to display the full set of information that is available to the Debugger in the small screen area shown. Therefore, the Debugger puts scroll bars onto Panes where there is more information than can be displayed on that part of the screen.

To see this additional information drag the thumb within the scroll bar or click on the arrows at either end of the scroll bar.

**Note:** Horizontal scroll bars are not automatically included in Expression, Disassembly or Source Panes but they can be added via **Ctrl + B**.

You can also scroll to the region you want by clicking on the required Pane to make it Active and then clicking and holding the left mouse button before dragging it to the top or bottom of the Pane.

**See Also:**  
**Changing Pane Sizes**

## Selecting A Pane Type

There are six types of Pane and you may display any number and combination.

A menu that allows you to change Pane properties is accessed via the Pane menu on the Menu bar or by right clicking the mouse on a relevant Pane. These menus are unique to the type of Pane that is Active but all the menus have the option **Change Pane** that allows you to switch between the different types.

Additionally, icons representing each type of Pane appear **adjacent** to the main toolbar.

**Registers Pane** - Displays the registers of the relevant CPU

**Memory Pane** - Displays areas of memory within the Target

**Source Pane** - Displays Source Files associated with program that CPU is running

**Disassembly Pane** - Displays the code that the CPU is running

**Watch Pane** - Displays 'watches' or expressions

**Local Pane** - Displays local variables

Click on the relevant icon to change the Active Pane.

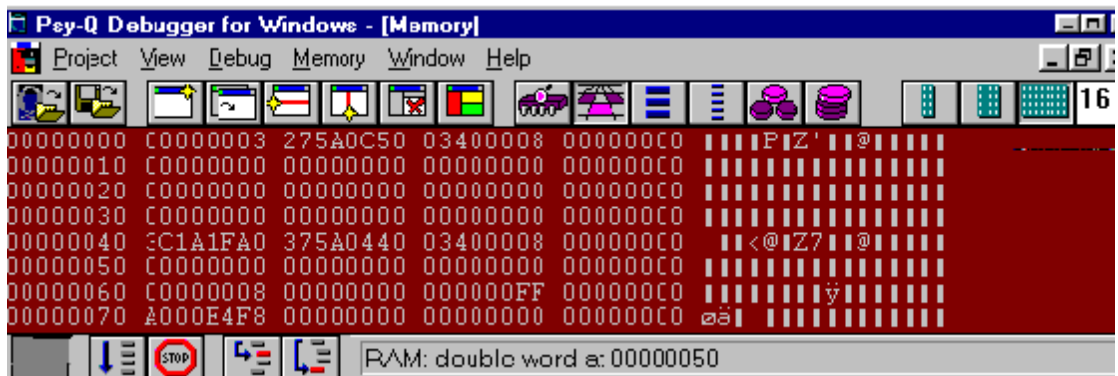
**Note:** You can also use the Hot Keys to switch between Pane types.

Icons representing menu options for the selected Pane are dynamically appended to the **far right** of the main tool bar. For example, if **Disassembly** Pane is Active, Disassembly Pane **options** will be displayed.

Further details about the options for each type of Pane can be found below.

## Memory Pane

There are three areas displayed on the Memory Pane: to the left is the memory address; in the middle is the value at the displayed memory address; and to the right is an optional ASCII display of the values which can be toggled on or off.



Memory Pane Display

You can **goto** an area of memory by typing the required address over the memory address or by selecting **Goto** from the Pane menu and entering a known address or label name to the dialogue box displayed.

**See Also:**  
**Moving To A Known Address Or Label**

Use the scroll bars or the **goto** functions described above to move around the display.

The default setting for the Pane is in bytes with the ASCII display set.

Change this default by selecting the Pane menu and choosing from the options:

- Bytes
- Words (bytes x 2)
- Double words (bytes x 4)
- ASCII (Toggle ASCII display on and off)
- Set Width (Changes the number of bytes displayed on a line)



Alternatively, clicking on these icons will activate the options listed above.

You can overtype the hexadecimal or ASCII displays to alter the content of the memory. A change to the hexadecimal display will be reflected in the ASCII display and visa versa.

When you move the mouse pointer over the values, the Status line displays the Memory Address and one of the following Memory Types:

- RAM
- ROM
- Invalid.

Invalid memory is displayed as question marks instead of hexadecimal values and full stops instead of ASCII.

The Set Width icon can be used to change the width of the display; click on the icon and type in the number of bytes to be shown on each line.

**Shift + Ctrl + F10** can be used to set the width of the Pane to match the size of the window.

The Active Pane can be made a **Memory Pane** via any one of the following methods:

Clicking on the Memory Pane icon on the toolbar: 

Using the Pane Type option from the Pane Menu

Using the Hot Key **Ctrl+Shift+M**

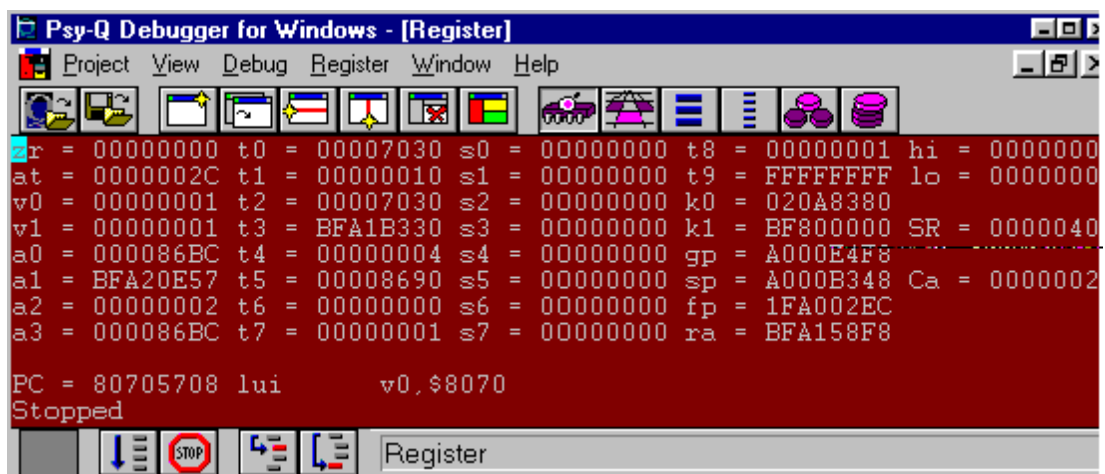
## Registers Pane

The Registers Pane shows the registers of the central processing Unit. These can be overtyped if required.

If the CPU has a Status Register, you can overwrite the individual bits by typing **R** to reset the bit or **1** or **S** to set it.

The display also shows the disassembled instruction at the Program Counter (PC) and the address of the instruction which will be executed next.

It also shows the current status and (if relevant) exception of the CPU on the bottom line of the Pane.



*Registers Pane Display*

When you click the right hand mouse button over a Registers Pane or select the Pane menu on the menu bar, you will see the **Change Pane Type** or **Pane Operations** options. Note that these are the only menu options for this type of Pane.

**Note:** If the current context stack level is **not 0**, the stack level (number) will appear in the Registers Pane as a visual warning.

### **See Also:**

**Using The Call Stack Display**

The Active Pane can be made **Registers Pane** via any of the following methods:

Clicking on the Registers Pane icon on the toolbar 

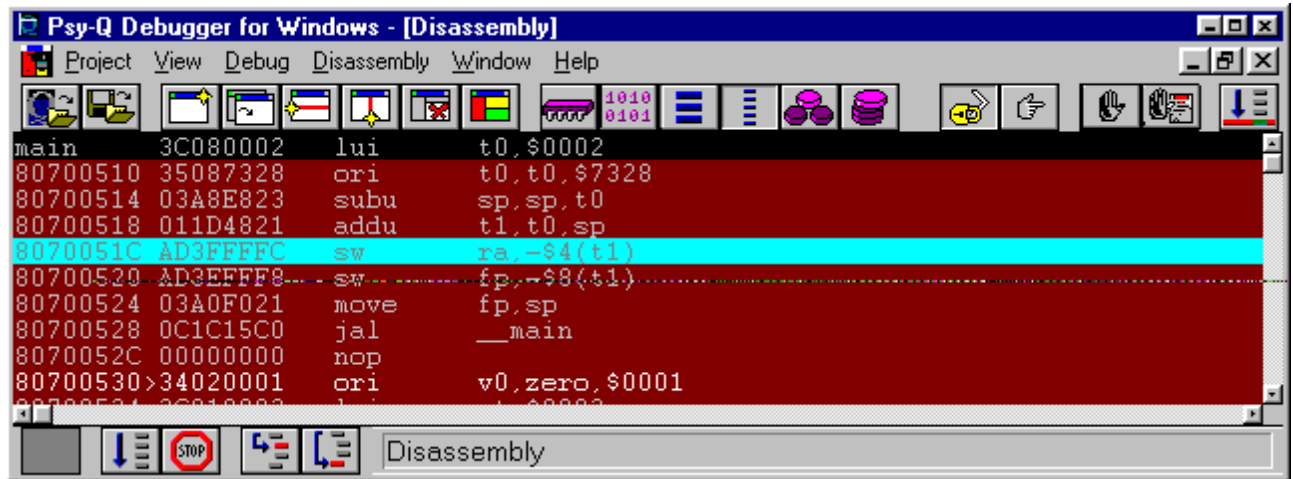
Using the Pane Type option from the Pane menu

Using the Hot Key **Ctrl+Shift+R**

## Disassembly Pane

The Disassembly Pane shows the disassembled code from an area of memory.

Four columns are displayed; the first shows the address or label; the second displays the values at that location in hexadecimal; the disassembled op code is shown in the third column and the fourth contains the op code parameters.



*Disassembly Pane Display*

When the cursor is positioned on a particular label on the Disassembly Pane, the relevant label name and value will be displayed on the Status line.

The Program Counter (PC) is shown on the screen preceded by the marker '>':

When you click the right hand mouse button over a Disassembly Pane or select from the Pane menu on the menu bar you see the following options:

- **Copy** to copy the address that the specified line represents, into the clipboard
- **Paste** to paste the specified address
- **Properties - Follow PC** to anchor the Pane to the Program Counter
- **Properties - Respond to Goto Breakpoint** to display breakpoints in all relevant Panes when you double-click a breakpoint
- **Properties - Centre on PC** to make the Registry global setting an individual property of the Disassembly Pane
- **Goto** to put the cursor at a known address or label name
- **Set PC** to set the PC to where the cursor is
- **Toggle breakpoint** to set and remove breakpoints
- **Edit breakpoint** to disable a breakpoint or make it conditional
- **Run to cursor** to run the Unit to the cursor position.




These options can also be activated by:

- Using the appropriate Hot Keys
- Clicking on these  icons

**Note:** **Ctrl + B** can be used to add horizontal scroll bars to the Disassembly Pane.

The Active Pane can become **Disassembly Pane** via any one of the following methods:

Clicking on the Disassembly Pane icon on the toolbar 

Using the Pane Type option from the Pane menu

Using the Hot Key **Ctrl+Shift+D**

**See Also:**

**Anchoring Panes To The PC**

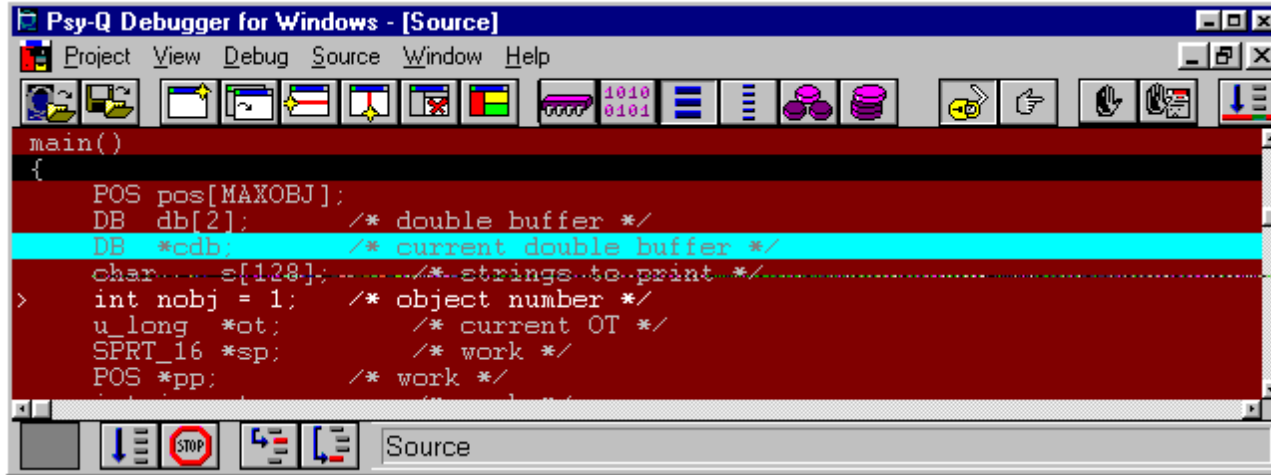
**Moving To A Known Address Or Label**

**Setting Breakpoints**

**Editing Breakpoints**

## Source Pane

A Source Pane displays one of the Source Files included in your Project.



*Source Pane Display*

When you click the right hand mouse button over a Source Pane or select from the Pane menu on the Menu bar you see the following options:

- **Copy** to copy the address that the specified line represents, into the clipboard
- **Paste** to paste the specified address
- **Properties - Follow PC** to anchor the Pane to the Program Counter
- **Properties - Respond to Goto Breakpoint** to display breakpoints in all relevant Panes when you double-click a breakpoint
- **Properties - Centre on PC** to make the Registry global setting an individual property of the Disassembly Pane
- **Goto PC** (space)
- **Goto** to put the cursor at a known address or label name
- **Source Files** to swap between the Source Files in the Project
- **Toggle breakpoint** to set and remove breakpoints
- **Edit breakpoint** to disable a breakpoint or make it conditional
- **Run to cursor** to run the Unit to the cursor position.

**Note:** If the Program Counter (PC) is at a line displayed on the Pane it will be preceded by the PC point line marker '>' and the line will be displayed in a different colour.

**Note:** If a breakpoint exists within the Pane it will display in a contrasting colour.

The options listed above can also be accessed:

- By using the appropriate Hot Keys

- By clicking on these  icons

**Note:** If the display is not set to follow the Program Counter (PC), the file displayed may not be the one executing at the PC.

**Note:** **Ctrl + B** can be used to add horizontal scroll bars to the Source Pane.

The Active Pane can be made **Source Pane** via any one of the following methods:

Clicking on the Source Pane icon on the toolbar 

Using the Pane Type option from the Pane menu

Using the Hot Key **Ctrl+Shift+S**

**See Also:**

**Anchoring Panes To The PC**

**Moving To A Known Address Or Label**

**Setting Breakpoints**


**Editing Breakpoints**

## Changing Source Files In The Source Pane

By default, the Source Pane displays the Source File which contains the PC or is blank if the PC is out of range of your source.



Any of the Project Source Files can be examined in this Pane by using the Source Files option from the Source Pane menu, as follows:

1. Select the Source Pane menu from the Menu bar.
2. Choose the Source Files option from the menu.
3. Select a Source File from the list displayed.
4. Click .

## Navigating Source Files In The Source Pane



Each Source Pane stores a history list of the Source Files which have been viewed in the Pane. The following keys are used to navigate this list:

<b>Ctrl + J</b>	To go backwards.
<b>Ctrl + K</b>	To go forwards.
<b>Ctrl + Shift + J</b>	Shows the previous Source File in the list but deletes the current Source File from the forward list.

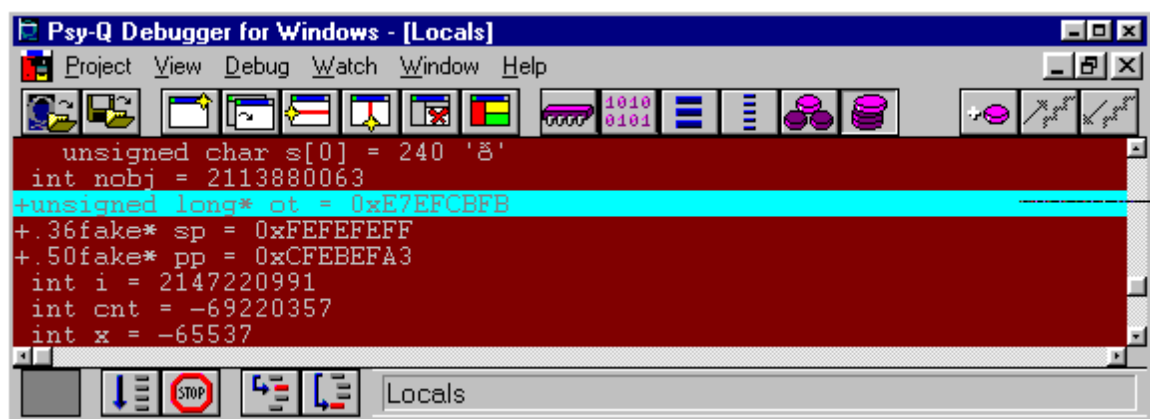
The history list is stored with the Pane in the View or Project. If you insert a new module into your Project (as opposed to adding one at the end), the history list may be transposed when you next load the Source Pane.

## Local Pane

The Local Pane is used to display all variables in the current local scope when you are debugging in C.

As you step and trace, the contents of this Pane will change to display the variables in the new scope.

You can expand or collapse variables and traverse array indices.



*Local Pane Display*

Variables can be viewed in hexadecimal or decimal modes by right-clicking within the Pane and 'toggling' between Hexadecimal/Decimal (on the displayed menu) as required. A tick will appear alongside Hexadecimal when this mode is selected.

**Note:** **Ctrl + Shift + T** will also toggle the **mode** between hexadecimal and decimal and **Ctrl + T** will toggle the mode for **single** expression.


Any Local variable that evaluates to a 'C', l-type expression, can be assigned a new value.

When you select the Local Pane menu or click the right hand mouse button over a Local Pane you see the following menu:

- **Expand/Collapse**- when the cursor is over a pointer, a structure or an array
- **Increase Index**- when the cursor is over an array element
- **Decrease Index**- when the cursor is over an array element.

These options can also be activated by:

- Using the appropriate Hot Keys

- Clicking on these icons 

**Note:** Use of the Local Pane is restricted to debugging in C.

The Active Pane can be made a **Local Pane** via any of the following methods:

Clicking on the Local Pane icon on the toolbar 

Using the Pane Type option from the Pane menu

Using the Hot Key **Ctrl+Shift+L**

**See Also:**

**Using The Call Stack Display**

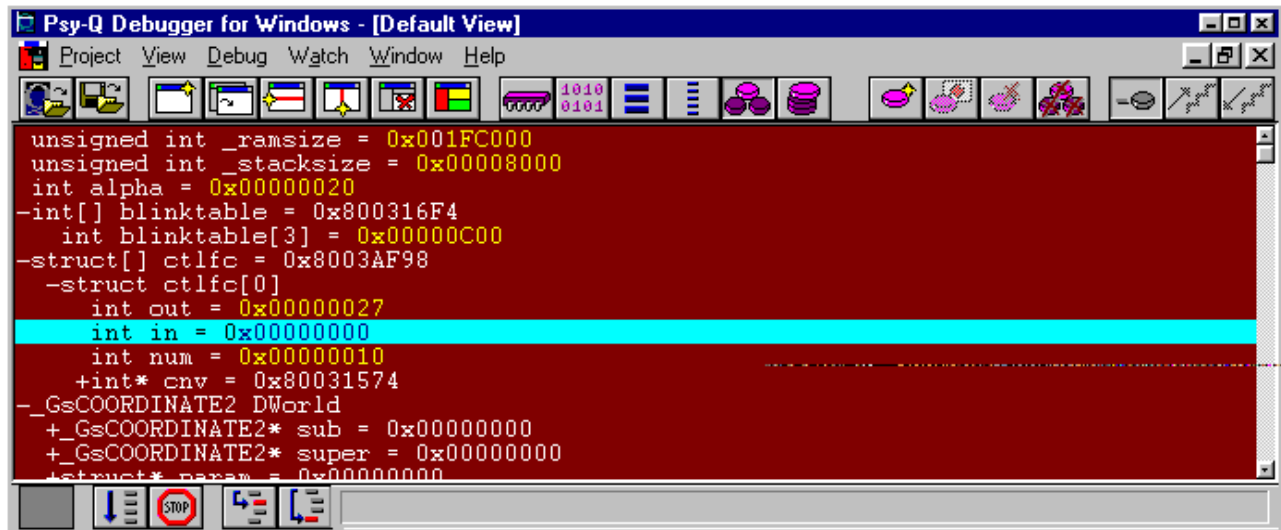
**Watch Pane**

**Expanding Or Collapsing A Variable**

**Traversing An Index**

## Watch Pane

The Watch Pane is used to evaluate and browse C type expressions.



*Watch Pane Display*

When you select the Watch Pane menu or click the right hand mouse button over a Watch Pane, the following menu is displayed:

- **Add Watch**
- **Edit Watch**
- **Delete Watch**
- **Clear All Watches**
- **Expand/Collapse**- to view/hide the components of a structure or an array
- **Increase Index**- to view higher indexed values within an array
- **Decrease Index**- to view lower indexed values within an array
- **Add Locals** - to add all local variables

**Note:** The AddLocals command will add all locals to the watch but unlike the Local Pane, when the watch goes out of scope the local variables will not be deleted.

These options can also be activated by the following methods:

- Using the appropriate Hot Keys
- Clicking on the appropriate icons

Structures, pointers and arrays can be opened in a Watch Pane.

- If you open a **structure** the members of that structure are displayed.
- If you open a **pointer** it is dereferenced.
- If you open an **array** the first element of the array is displayed.

The contents of the Watch Pane are saved within the View when the Project is saved.

Variables can be viewed in hexadecimal or decimal modes by right-clicking within the Pane and 'toggling' between Hexadecimal/Decimal (on the displayed menu) as required. A tick will appear alongside Hexadecimal when this mode is selected.

**Note:** **Ctrl + Shift + T** will also toggle the **mode** between hexadecimal and decimal and **Ctrl + T** will toggle the mode for a **single** expression.

Any Watch variable that evaluates to a 'C', l-type expression can be assigned a new value.

**Note:** The options Expand/Collapse and Increase Index '+' and Decrease Index '-' are only available for arrays, pointers and structures.

### **See Also:**

**Using The Call Stack Display**

**Hot Keys**

**Assigning Variables**

**Expanding Or Collapsing A Variable**

The Active Pane can be made a **Watch Pane** via any one of the following methods:

Clicking on the Watch Pane icon on the toolbar 

Using the Pane Type option from the Pane menu

Using the Hot Key **Ctrl+Shift+W**



## C Type Expressions In Watch Pane

The following 'C' type expressions, shown in order of precedence, may be used to evaluate expressions within the Watch View of a Project:

[ ]	array subscript
->	record lookup
~ - * &	unary prefix
* / %	multiplicative
+ -	additive
<< >>	bitwise shifting
<> <= >=	comparatives
== !=	equalities
&	bitwise and
^	bitwise xor
	bitwise or


**Note:** As in C, parenthesis can be used to override precedence.

## Assigning Variables

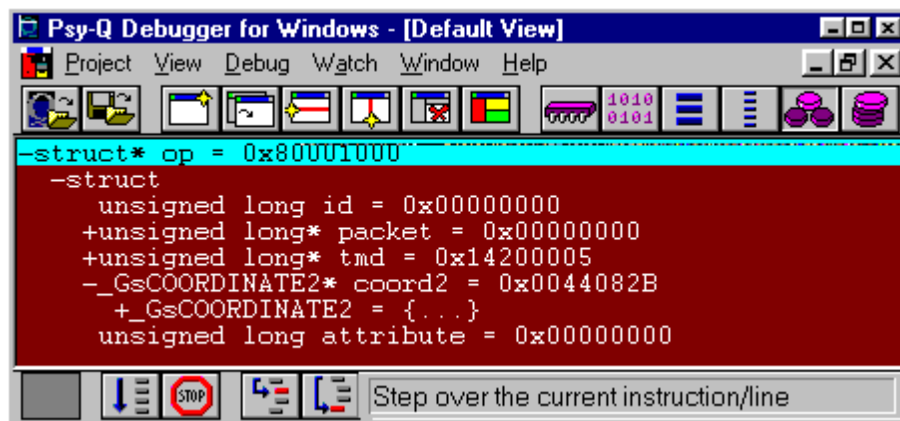
Any variable that evaluates to a 'C', l-type expression can be assigned a new value. For example, in the case of a de-referenced pointer, a new value can be assigned to the pointer or the de-referenced expression.



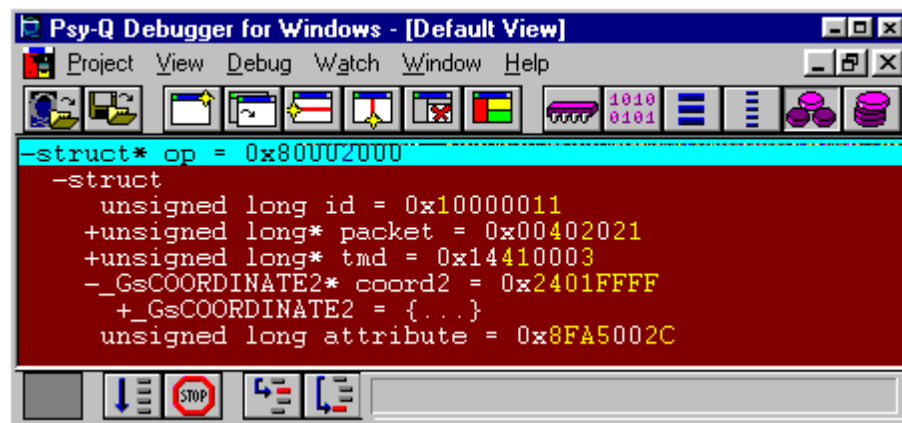
Variables are assigned as follows

1. Place the caret over the required expression to make it Active.
2. Press '='.
3. Enter the new value to the displayed dialogue box; this can be another expression if required.
4. Click .

In the example below, this facility was used to assign a new value **0x80002000** to the specified pointer. The de-referenced structure changes to reflect the amended value.



*Displayed Structures For Pointer Address*



*Amended Structures After Pointer Assigned New Variable*

**IMPORTANT:** The expression that you are assigning and the new value **must** have compatible types.

**Note:** Variables can be assigned whilst the Target is running.

## Expanding Or Collapsing A Variable

Pointers, structures and arrays are variables which can be expanded or collapsed in the Local or Watch Panes and the Call Stack Display when you place the caret over them.

If you expand a **pointer** a line will be added below for the dereferenced pointer. For example if the pointer is to an integer, the dereferenced pointer will display that integer.

An expanded **structure** will display all the elements of that structure below it.

For an expanded **array** the second line of the display will display the first element of the array.



To expand or collapse a variable:

1. Select the Pane menu for the Local or Watch Panes.
2. Choose the Expand or Collapse option from the menu.


When shown in the Watch Pane, expressions which can be expanded or collapsed will be prefixed as follows:

+	this indicates an expression that can be expanded
-	this indicates that the expression is expanded and can be closed.

This is followed by the expression's type and value.



To edit an expression, Shift +double-click or highlight it and press Return.

**Note:** It is also possible to expand or collapse an expression by using the expand or collapse icons  on the Pane toolbar or pressing SPACE.

**Note:** Ctrl+SPACE or Ctrl+double-click will **expand all** the elements in an array.

## Traversing An Index

You can traverse an index if the caret is on an array element in the Local or Watch Panes.


If an index is increased, the array will display **the next** array element.

Decreasing an index causes **the previous** array element to be displayed.



To increase or decrease an index:

1. Select the Pane menu for the Local or Watch Panes.
2. Choose the Increase Index or Decrease index option from the menu.

**Note:** It is also possible to expand or collapse a variable by using the increase index or decrease index icons  on the Pane toolbar.

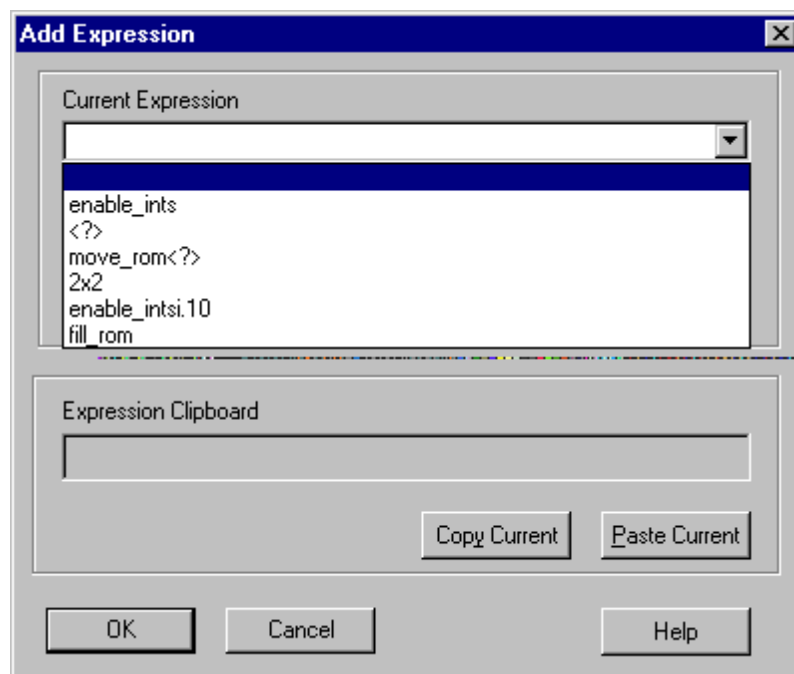
## Adding A Watch

The Watch Pane is used to evaluate and browse C type expressions.



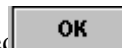
To add a watch or expression

1. Make the Watch Pane the Active Pane.
2. Select the Watch Pane menu from the Menu bar.
3. Choose the Add Watch option from the menu.
4. Type the required expression directly into the Add Expression dialog box or click the down arrow to display expressions which have been used previously.




*Add Watch Dialogue Box*

5. Enter or click the required expression and select



The Debugger also offers various ‘matching’ facilities whereby you can enter a partial value and the program will search the current and global scopes for those matching the specified criteria or browse all available symbols. These are described below in **Using The Expression Manager**


**Note:** It is also possible to add a watch by clicking on the add watch icon  on the Watch Pane toolbar or using the Hot Key **Insert**.


**See Also:**

**Using The Expression Manager**

## Editing A Watch

Any of the C type expressions that you can enter into the Watch Pane can be edited as follows:

1. Make the Watch Pane the Active Pane.
2. Select the Watch Pane menu from the Menu bar.
3. Choose the Edit Watch option from the menu.
4. Select the watch to edit.
5. Amend as necessary via the Edit Expression dialog and click . History, browsing and matching facilities are available via this dialogue box.

**Note:** It is also possible to edit a watch by clicking on the Edit Watch icon  on the Watch Pane toolbar.

**Note:** To view variables in hexadecimal, right-click within the Pane and 'toggle' 'Hexadecimal/Decimal' as necessary. A tick will appear alongside Hexadecimal when this option has been selected. **Ctrl + Shift + T** can also be used to toggle the mode displayed. You can also toggle the mode for a **single** expression via **Ctrl + T**.

**See Also:**

**Using The Expression Manager**


**Previously Entered Expressions History List**

## Deleting A Watch



Any of the C type expressions entered into the Watch Pane can be deleted as follows:

1. Make the Watch Pane the Active Pane.
2. Select the Watch Pane menu from the Menu bar.
3. Choose the Delete Watch option from the menu.
4. Select the watch and press Enter.

**Note:** It is also possible to delete a watch by clicking on the Delete Watch icon  on the Watch Pane toolbar or pressing DEL.


**Note:** You can only delete a Watch at the root of the expression, not on any expanded part of it.

## Clearing All Watches



All of the C type expressions entered into the Watch Pane can be removed in one action, as follows:

1. Make the Watch Pane the Active Pane.
2. Select the Watch Pane menu from the menu bar.
3. Choose the Clear All Watches option from the menu.

**Note:** You can also clear all watches by clicking on the Clear All Watches icon  on the Watch Pane toolbar.

## Debugging Your Program

The Debugger helps you to detect, diagnose and correct errors in your programs. This is achieved via facilities which enable you to step and trace through your code in order to examine local and global variables, registers and memory.

Breakpoints can be set wherever you need them at C and Assembler level and if required, these breaks can be made conditional on an expression. Additionally, selected breakpoints can be disabled for particular runs.

Your choice of Views depends on the level at which you are debugging. For example it is appropriate to use a Register Pane for assembler debugging and a Local Pane when debugging in C.



## Specifying The Polling Rate And Continual Update Rate

It is possible to set the rate at which the Debugger checks the exception state of the Target, i.e. determine the granularity of discovering an exception when the Target is running.

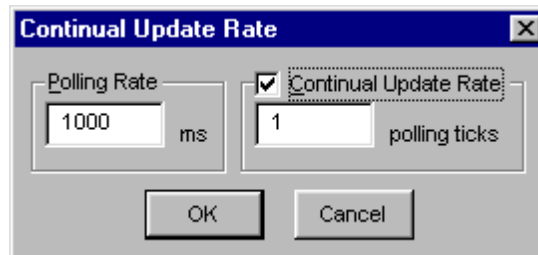
**Note:** Even though checking the exception state has zero hit for some Targets, it is not recommended that you set this value below **250 ms**.

It is also possible to specify the rate at which the Debugger updates information while the Target is running. This is particularly important for Targets which connect independently of a pollhost( ) since rapid connection rates may cripple the Target. It is measured in polling ticks, e.g. 250 x 4 ticks = a continual update rate of 1 second. If required, continual update (not polling) can be turned off altogether.




The rates are set as follows:

1. Select ContinualUppdate Rate from the Project option on the main menu or press **Ctrl+I**. A dialogue box displays the current polling rate in milliseconds:



*Update Rate Dialogue Box*


2. If required, enter a new value and select . The rate is saved between all debugging sessions and not as part of a Project.
3. The continual update rate is measured in polling ticks; enter a new value if required.



To turn off the continual update rate:

De-select the Continual Update Rate; only the polling rate will still be active.

## Forcing An Update

During continual update, the information you see in the Debugger windows won't be updated until the next connection; therefore, the slower the update rate, the longer it will be before exceptions can be spotted. However, it is possible to force an update by pressing **Ctrl+U**, selecting the **Update** option from **Debug** on the main menu or clicking the  button on the toolbar.

## Using The Expression Manager

The Expression Manager Dialog is used to enter an expression, for example to add a watch to the Watch Pane or specify a location for a breakpoint.

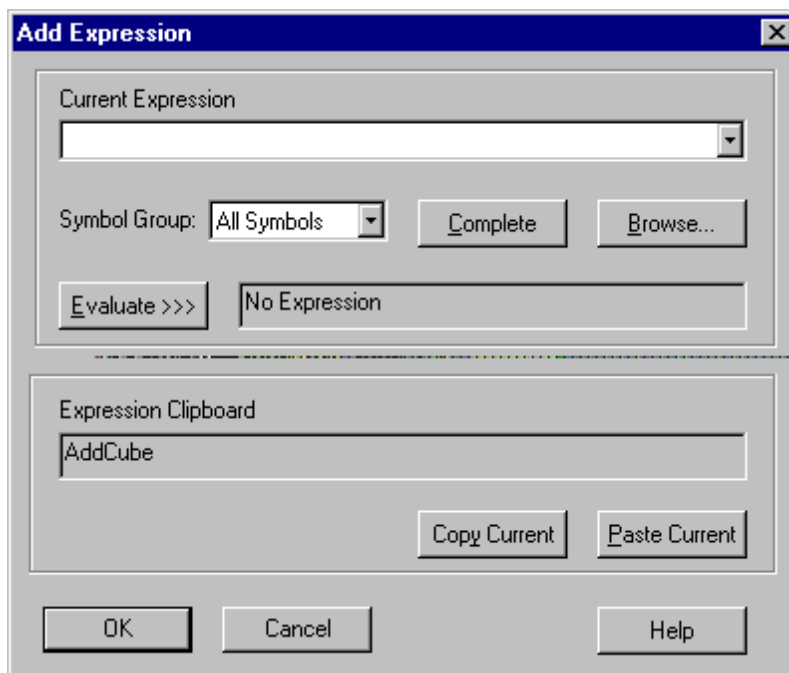
In summary, the Dialog provides the following facilities:

- A history list of previous expressions
- Name completion
- Enhanced browsing features
- The ability to select from various groups of symbols
- Expression copying and pasting to/from the expression clipboard
- Instant expression evaluation



To access the Expression Manager:



The dialog will be displayed whenever you select an option which requires you to specify an expression. The title bar will reflect the context from which it was called.

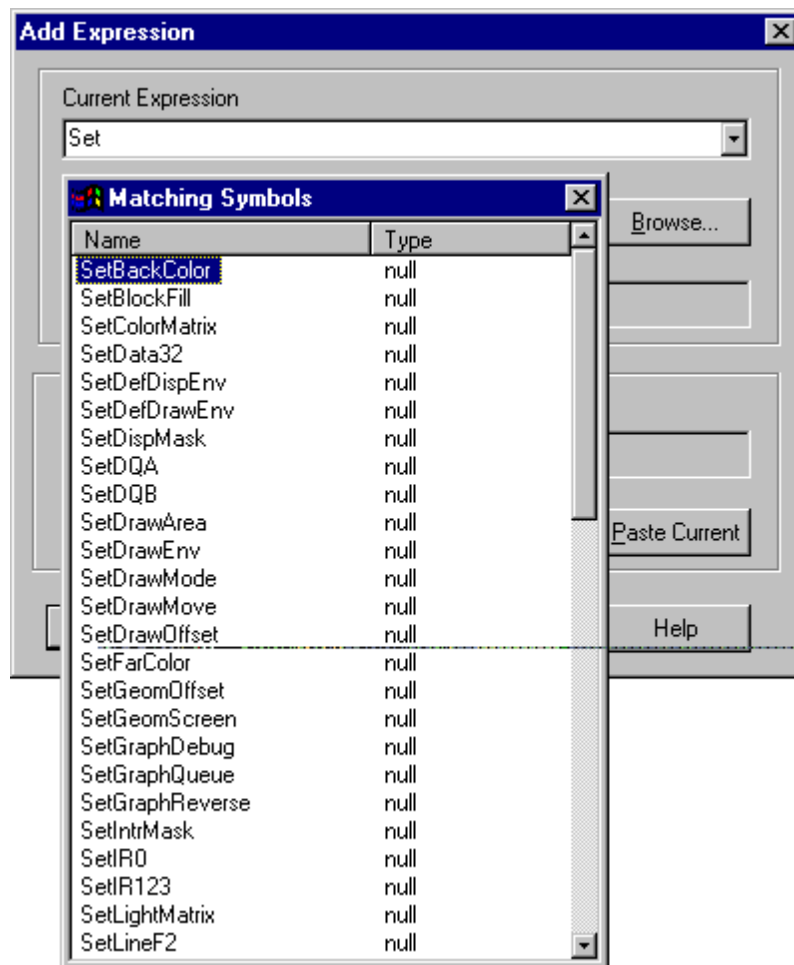


*Expression Manager Dialog*



To specify an expression:

1. Enter text or symbols to the **Current Expression** box.
2. If you do not know the format of the required expression, click  (or press **Alt+B**) and the **Symbol Browser** will display all symbols (and their details) in the current scope. See **Using the Symbol Browser** below.
3. A pull-down history list of the most recently entered expressions is also available from the **Current Expression** box.
4. Select or specify as appropriate.
5. Alternatively, if you know and have entered part of the expression, click  (or press **Alt+C**) and use the **name completion** facility; this will take the text fragment from the left of the cursor and attempt to match it with the start of the debugging symbols found in the current scope. If **single** match is found the text is replaced by the complete symbol. Where more than one match is found a separate window will show the matching symbols in alphabetical order.



*Expression Manager Displaying Matching Symbols*

6. Select the symbol to be inserted into the expression; press ESC to cancel the operation.
7. By default, **all** available symbols are matched during name completion, however, you can change this to match **only** symbols of a specific group by selecting an alternative from the **Symbol Group** list. The selection can be restricted to any of the following groups:

<b>All Symbols</b>
<b>Globals</b>
<b>Locals</b>
<b>Functions</b>
<b>Types</b>
<b>Assembler Labels</b>

**Note:** The group selected here will also be the first one displayed by the **Symbol Browser**. See point 2 above and **Using the Symbol Browser** below.

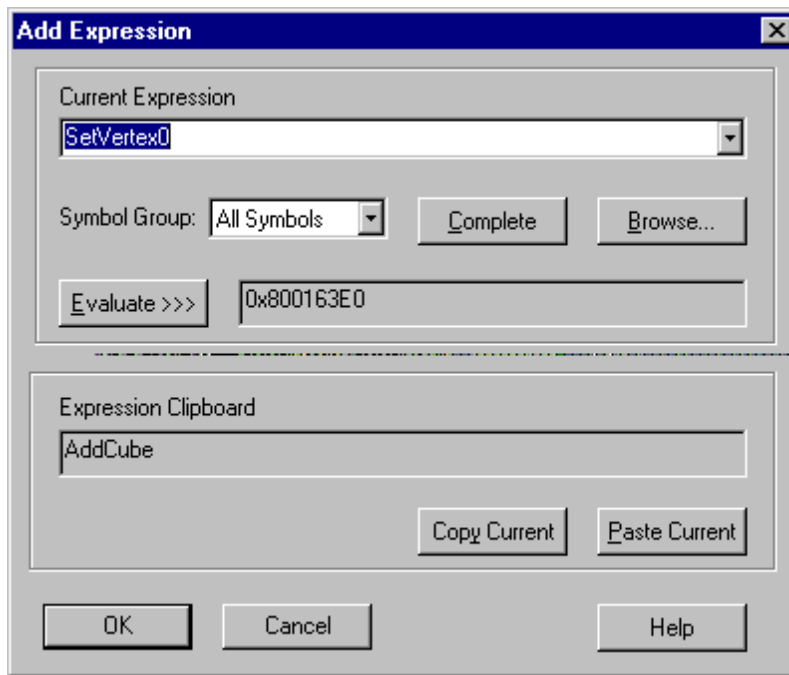
8. Click  when your expression is complete; it will be inserted into the relevant Pane.



To evaluate an expression:

At any time you can attempt to evaluate the current expression by clicking

(or pressing **Alt+E**). If the expression can be evaluated, its memory address will be displayed in the adjacent window, otherwise a suitable error will be given.



*Expression Manager Displaying Evaluated Expression*



To use the Expression Clipboard:


1. Click **Copy Current** (or press **Alt+Y**) to copy the current expression into the Debugger's Expression Clipboard; this can be recalled later within other Debugger components

**Note:** The expression text is also copied into the Windows Clipboard to be used by other Windows applications.

2. The contents of the Expression Clipboard can also be inserted into the current expression by clicking **Paste Current** or pressing **Alt+P**.




To use the Symbol Browser:

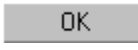
1. Click  (or press **Alt+B**) and the **Symbol Browser** will display all symbols (and their details) in the current scope.

The details are as follows:

<b>Name</b>	The name of the symbol as it would appear in your code.
<b>Type</b>	For storage variables - type (eg int, float); For functions - return type.
<b>Class</b>	'C' class of the symbol, e.g. automatic, static, union, typedef etc.
<b>Location</b>	The memory location or position relative to the stack the symbol refers to, if applicable to the symbol.
<b>Dims</b>	The number of dimensions of the symbol if it represents an array and also the size of each of its dimensions.
<b>Tag</b>	The structure or enumerator tag if applicable.

2. You can restrict the number of symbols which will be displayed by selecting a particular group of symbols from the pull-down menu and clicking .
3. In addition, the 'wildcard' matching facility can be used in conjunction with the specified symbol group to restrict the display to particular symbols. Enter wildcard text in the edit box. A '\*' character will match 0 to any number of text characters, while a '?' character will match any single text character. All other characters will only match the equivalent text character. 'm\*?n' will match all symbols in the specified symbol group that start with 'm', end in 'n' and have at least one character between, for example 'main'.

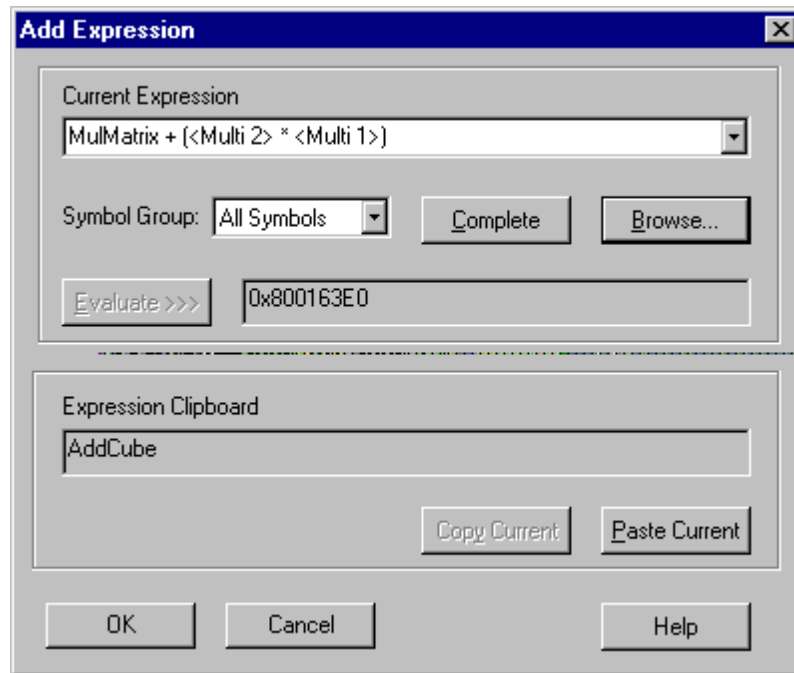
By default, all symbols are sorted and displayed in alphabetical name order. However, if you click on the heading name for any of the details columns, the symbols will be sorted by the relevant column heading.

4. Click or press RETURN on a symbol name and it will be inserted into the **Current Expression** box in the Expression Dialog.
5. Click  to insert the expression in the relevant Pane.



To specify multiple expressions:

1. If required, more than one expression and symbol can be entered in the Current Expression box.
2. Use the mouse and Ctrl and/or Shift keys to make a specific selection and then press RETURN to insert it in the **Current Expression** box; the expressions will be represented by appropriate text in angled brackets '<Multi 1>' and '<Multi 2>' as shown below:

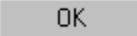


*Add Expression Dialog Displaying Representation Of Multiple Expressions*

Up to three multiple selections can be specified at a time.

Single expressions can be specified alongside multiple groups.

If a group is deleted in the **Current Expression** box, **all** symbols and expressions within the group will be removed.

3. When completed, click  to insert the expression(s) in the relevant Pane.



## Setting Breakpoints

Breakpoints can be set in Source and Disassembly Panes; they appear in the Pane as a different coloured bar.

A Project can have many breakpoints set and they are saved when the Project is saved. They are restored relative to Assembler labels wherever possible; this ensures they are preserved even when you alter the source code and rebuild.

The Debugger supports a Pane and Tool based Breakpoint Plug-In.

**The Breakpoint Manager Pane and Breakpoint Manager Dialog** allow you to view all current breakpoints. They also allow you to set breakpoints anywhere in memory and alter a breakpoint's type and settings.

There are four types of breakpoint:

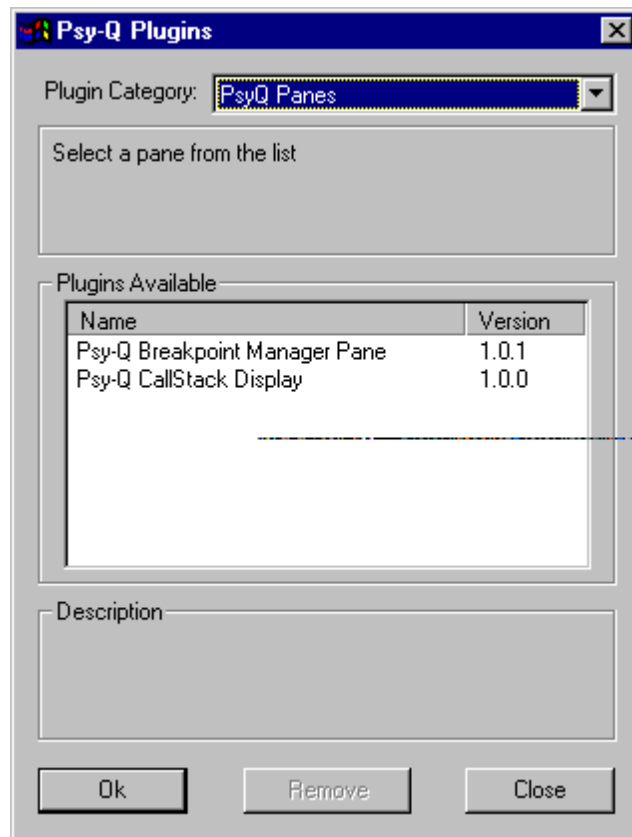
- |                    |   |
|--------------------|---|
| <b>Absolute</b>    | The PC stops when it arrives at the breakpoint.   |
| <b>Counter</b>     | The counter is increased by one every time the PC passes the breakpoint. This does not stop the PC.   |
| <b>Countdown</b>   | The counter is decreased by one from an initial user-set value every time the PC passes the breakpoint. When the counter reaches zero the PC stops at the breakpoint. |
| <b>Conditional</b> | The PC stops at a conditional breakpoint only if a user-specified expression evaluates as TRUE.   |



To access the Breakpoint Manager Pane:

1. Right-click within a Source or Disassembly Pane and select from the displayed sub-menu. Left-click within the empty Pane.

The Plug-In Manager will display the currently installed Plug-Ins.



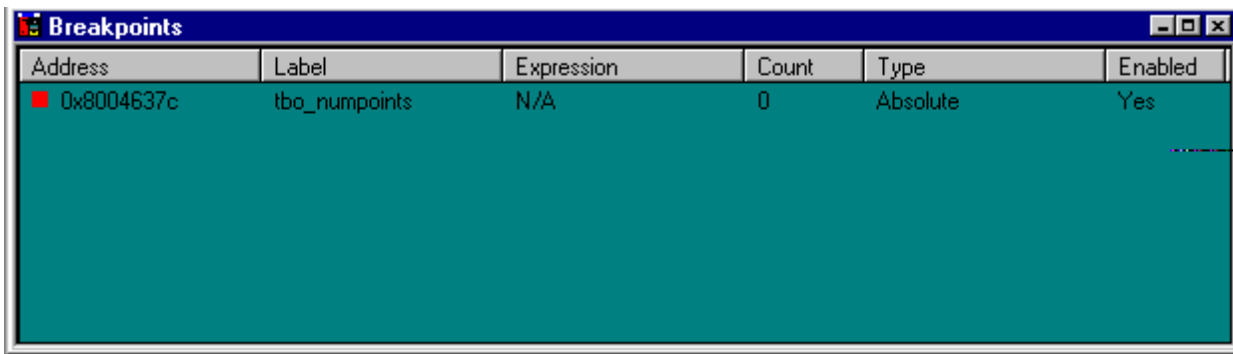
*Plug-In Manager*

2. Select the Breakpoint Manager Pane and click .

The selected Pane displays a list of the breakpoints which are currently set. Where relevant the following information is shown for each one:

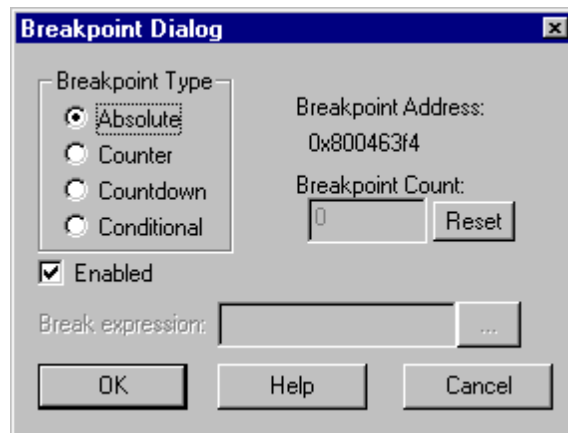
<b>Address</b>	The memory location of the breakpoint.
<b>Label</b>	The nearest label to the breakpoint.
<b>Expression</b>	Any expression used to create a conditional breakpoint.
<b>Count</b>	The number of steps since the PC passed the breakpoint or until the breakpoint is reached.
<b>Type</b>	The breakpoint type.
<b>Enabled</b>	The current state - Yes or No.

The breakpoint's type is also reflected in the icon displayed next to its address. This icon changes to a 'triggered' icon when the breakpoint stops the PC.




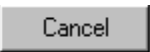

*Breakpoint Manager Pane*

- ▶ To access the operations menu for a breakpoint:  
Right-click over the required breakpoint.  
A sub-menu is displayed, from which you can carry out the following actions:
  - ▶ To add a breakpoint:  
In the displayed Expression Manager enter the memory address where you wish the breakpoint to appear. See **Using The Expression Manager** for further details.
  - ▶ To remove a breakpoint:  
Toggle the option for the selected breakpoint.
  - ▶ To disable a breakpoint:  
Toggle the option for the selected breakpoint.
  - ▶ To alter breakpoint properties:
    1. The Breakpoint Dialog will display the current properties for the selected breakpoint.



2. Change the **Breakpoint Type** if required. If *Countdown* is chosen you must enter a value to the **Breakpoint Count** box.

You must specify a **Break expression** if the **Breakpoint Type** is *Conditional*. Use the adjacent browse button to search for available symbols.

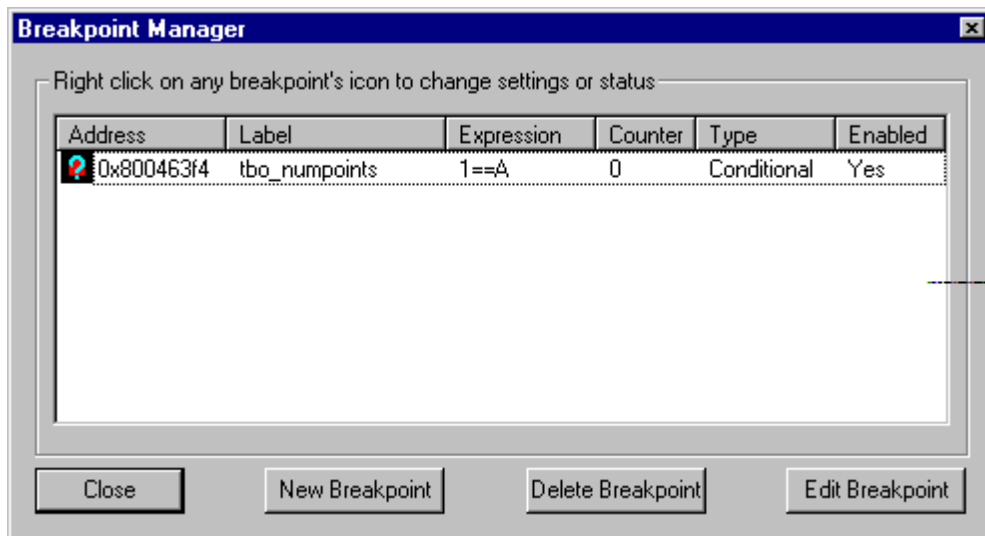
3.  can be used to reset the Breakpoint Count to zero when *Counter* has been selected.
4. De-select or select the **Enabled** box if required. When this check box is set the breakpoint is enabled and only these will be included in a debugging run.
5. Use  to leave the dialogue box without saving the changes you have made.
6. Click  when the amendments are complete.

**Note:** This dialog is also accessible via the Edit Breakpoint option from the Source or Disassembly Panes.



To access the Breakpoint Manager **Dialog**:

From the main **T**ools menu, select the Breakpoint Manager Dialog. This performs identical functions to the Pane but uses buttons as well as the right-click menu. It provides a quick means of displaying and amending breakpoint settings without the necessity of switching Pane types.





It is also possible to directly set or edit a breakpoint as follows:

1. Make a Source or Disassembly Pane Active.
2. Click on the instruction or line at which you want to set the break or amend an existing one.
3. Select the Source menu from the Menu bar.
4. Choose the Toggle or Edit breakpoints option from the menu.

**Note:** An **Absolute** breakpoint will be inserted by default when the Toggle breakpoints option is used. Use Edit breakpoints and a relevant selection to insert any other type.

A breakpoint can be removed by clicking on the colour bar and toggling the menu option taken to create it.

**Note:** Breakpoints can also be set and removed via the **F5** key or the set / unset breakpoint icons on the Pane toolbar .


**Note:** The  button on the toolbar will clear all breakpoints, Note however that if you have upgraded a previous beta version, this button will not be available unless you have customised the toolbar.

Any of the following methods can be used to create, remove and edit breakpoints:

The **F5** key

**Shift + F5** or **Shift + Double-click** will display the Breakpoint Dialog from which breakpoints can be edited.

The Breakpoint options from the Source or Disassembly Pane menus

The Breakpoint icons from the Pane toolbar .

The View and Edit Breakpoints option from the Tools menu.

**Note:** Double-clicking on a breakpoint in a Source and Disassembly Pane will goto that breakpoint. De-selecting the **Respond to breakpoint event** in the Pane's context menu however, will switch this property off. The information is saved with the View or Project.

## Using The Call-Stack Display

The Call-Stack Display Pane Plug-In allows you to view the whole of the stack and step up and down it to a particular context.

The benefit of this is that it allows you to see the functions being called and where they were called from. It is also possible to see the parameter types and values that were passed to a function.

**Note:** The Debugger will only provide call stack information when it detects there is a valid frame pointer and there are symbols available for the current function.

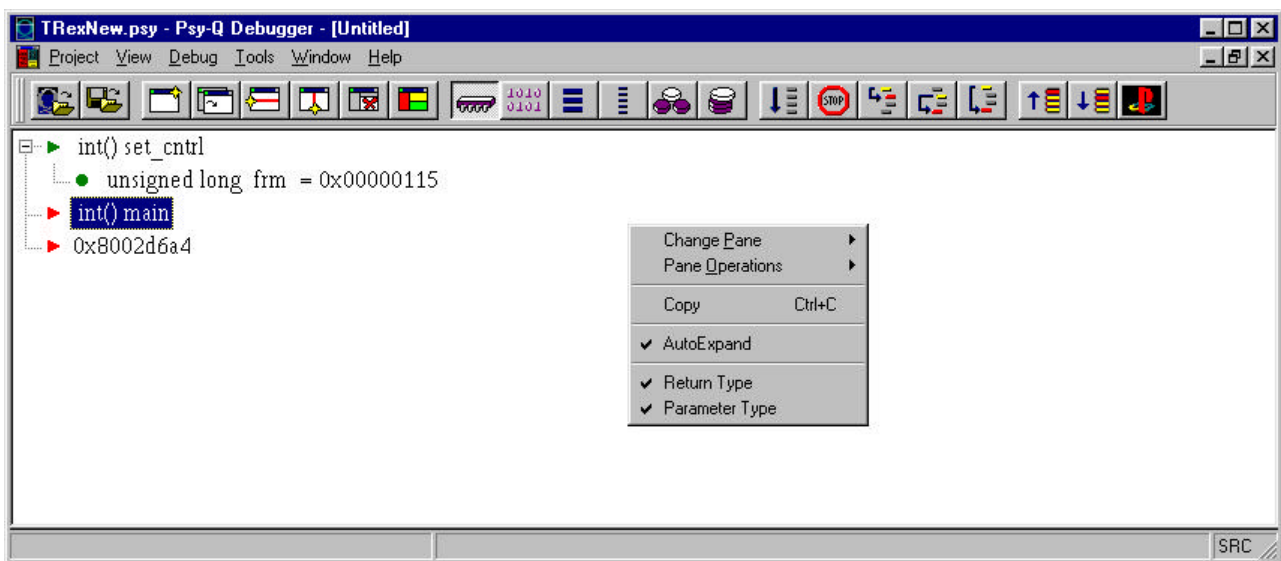


To display the Call Stack:

1. Right-click within any Pane and select **Other** from the displayed sub-menu. Left-click within the empty Pane.

The Plug-In Manager will display the currently installed Plug-Ins.

2. Select the Call-Stack Display Plug-In and click .



*Call-Stack Display*

The Call-Stack Display dialog provides a view of the whole stack; an icon (a green triangle) points to the current context. Other contexts are designated by a red triangle. Parameter details are highlighted by a red or green circle depending on whether they are part of the current context or not.

The Windows 95 tree control functions have been used in this dialog to show information about the stack. You can therefore quickly see which context the stack is in and which function calls have been placed on the stack. You can expand a branch if you wish to view more information about a function call. If they are available, branches will be shown for function return values, parameter types or parameter values.



To use the scroll bar:

1. A scroll bar on the right of the tree control allows you to view various portions of the stack.
2. When you click on a stack context, the pointer will move to that context and reposition the view on the stack accordingly.



To switch contexts:

1. You can alter the context (machine state) to reflect a previous function call by clicking on a particular context in the tree control.
2. The green icon will be repositioned to reflect that the machine's current state has been changed and other windows will be updated accordingly.

**Note:** You can customise the tool bar to include the following stack stepping icons:



- To increase the stack level context.



- To decrease the stack level context.



To expand a branch:

1. Right-click within the Pane.
2. A pop-up menu allows you to display return values, parameter types or parameter values. Left-click an item to select it and a check mark will appear alongside it. Left-click again to de-select the option.
3. If a cross is shown to the left of displayed parameters, this means that the parameter is a structure, pointer or an array. In the same way that parameters can be expanded in a watch window, clicking on this cross will also reveal the derived types and values.

**Note:** When the **autoexpand** menu item is selected, parameter details associated with a context will be displayed, provided that at least one parameter detail option is selected.





To copy a selection to the Expression Clipboard:

1. It is possible to select an item from the Call Stack Display and copy it into the Expression Clipboard for subsequent use in a different application.
2. Select the item to be copied.
3. Press **Ctrl + C** or right-click and select the **Copy** option from the displayed pop-up menu.

**Note:** The selected expression also appears on the Debugger's status bar.



To change properties of the Call Stack:

1. Right-click within the Pane.
2. A pop-up menu allows you to display return values, parameter types or parameter values. Left-click an item to select it and a check mark will appear alongside it. Left-click again to de-select the option.

**Note:** Properties selected via the right-click pop-up menu are saved when the Project is saved.

## Stepping Into A Subroutine

The Step Into command allows you to trace the execution of the program one step at a time and so isolate any bugs that might be present.

When you Step Into a subroutine call, the Program Counter moves to the start of the subroutine and displays the relevant code. At the end of the subroutine you will be returned to where it was called from.

At Assembler level a debugging step is the execution of a single instruction.

At **Source** level, one line at a time will be executed in each step and any subroutines or calls within that line will be stepped into.


The current stepping mode is indicated by **SRC** or **DIS** on the far right of the status bar; the Debugger will change this to the appropriate mode when either a Source or Disassembly Pane become active. You can override this however by using **Ctrl + M** to toggle between the two.

**Note:** If the mode is SRC and a Source level step cannot be performed, a Disassembly level step will be performed instead.



To Step Into a subroutine during debugging:

1. Select the **D**ebug menu from the Menu bar.
2. Choose the **S**tep **I**nto option from the menu.

**Note:** Alternative ways of Stepping Into a subroutine are to use the Step Into icon on the Unit toolbar (at the bottom of the Debugger window  or to press **F7**. Note that it is possible to use the Step Into icon for a non-Active View.

## Stepping Over A Subroutine

When you use the Step Over command, the subroutine is executed but not displayed and the Program Counter moves to the next line of calling routine code.

At Assembler level a debugging step is the execution of a single instruction.

At **Source** level, one line at a time will be executed in each step and any subroutines or calls within that line will be performed.


The current stepping mode is indicated by **SRC** or **DIS** on the far right of the status bar; the Debugger will change this to the appropriate mode when either a Source or Disassembly Pane become active. You can override this however by using **Ctrl + M** to toggle between the two.

**Note:** If the mode is SRC and a Source level step cannot be performed, a Disassembly level step will be performed instead.



To Step Over a subroutine:

1. Select the Debug menu from the Menu bar.
2. Choose the StepOver option from the menu.

**Note:** Alternative ways of Stepping Over a subroutine are to use the Step Over icon  by the Unit menu or to press **F8**. Note that you can use the Step Over icon for a non-Active View.

## Stepping Out Of A Subroutine

The Step Out Of command returns you to the command after the statement that called the current function, i.e. it unpicks the stack one level, calculates the Program Counter at this level, sets a temporary breakpoint at this address, sets the stack level to **0** and then runs the target.

**Note:** In order to use the Step Out command, the Debugger requires full symbol information for the function you are in and you must not be executing prolog or epilog code.



To Step Out of a subroutine:

1. Select the Debug menu from the Menu bar.
2. Choose the Step Out option from the menu.

**Note:** Alternative methods of Stepping Out of a subroutine are to click on the Step Out Of icon on the Pane toolbar  or to use the Hot Key **F12**.

## Running To The Current Cursor Position


The Run to Cursor command can be used during debugging within the Source and Disassembly Panes.



To run to the current cursor position:

1. Make a Source or Disassembly Pane active.
2. Click on the displayed code at the point you want to run to.
3. Select the Source or Disassembly menu from the Menu bar.
4. Choose the Run To Cursor option from the menu.

If the Unit does not reach the cursor position it will continue running.

**Note:** Alternative methods of running to the cursor are to click on the Run To Cursor icon on the Pane toolbar  or to use the Hot Key **F6**.

**Note:** You can use Run To Cursor while the Unit is running to make it stop at the cursor position.

## Running Programs

The Run command causes the CPU of the specified Unit to start running.


It will continue until it meets a breakpoint, a processor exception or is stopped by the Stop or Run To Cursor commands.

During a debugging run the various Panes will show the progress of the run.



To start the program running

1. Select the Debug menu from the menu bar.
2. Choose the Go option from the menu.

**Note:** Alternative ways to start the run are to click the Start button on the relevant Unit toolbar  or to press **F9**. **F9** will also stop the Target if it is running.


## Stopping A Program Running

The Stop command halts the CPU of the specified Unit as soon as possible.



It is specified as follows

1. Select the Debug menu from the Menu bar.
2. Choose the Stop option from the menu.

**Note:** Alternative ways to stop the run are to click the Stop button on the relevant Unit toolbar  or to press **Esc**.

## Moving The Program Counter

The program counter (PC) can be set via the Set PC command.

This command moves the program counter to the current cursor position.



It is found on the Pane menus for Source and Disassembly Panes and is set as follows

1. Make a Source or Disassembly Pane Active.
2. Place the caret where you wish the PC to move to.
3. Click the right hand mouse button to call the Pane menu.
4. Select the Set PC option from the menu.

With this command, no instructions are executed between the previous and new PC position.

The opposite command to Set PC is Goto PC which takes the cursor to the position of the Program Counter.

**Note:** An alternative way to activate the Set PC command is by using the Hot Key **Shift+Tab**.

## Moving The Caret To The PC

The caret point can be placed at the program counter address via the Goto PC command.

This is found on the Pane menus for Source and Disassembly Panes.



To Set The PC:

1. Make a Source or Disassembly Pane Active.
2. Click the right hand mouse button to call the Pane menu.
3. Select the Goto PC option from the menu.

Goto PC is the opposite command to Set PC which sets the Program Counter to the value at the current caret position.

**Note:** Alternatively, pressing the 'space' bar will **directly** place the caret point at the program counter address.

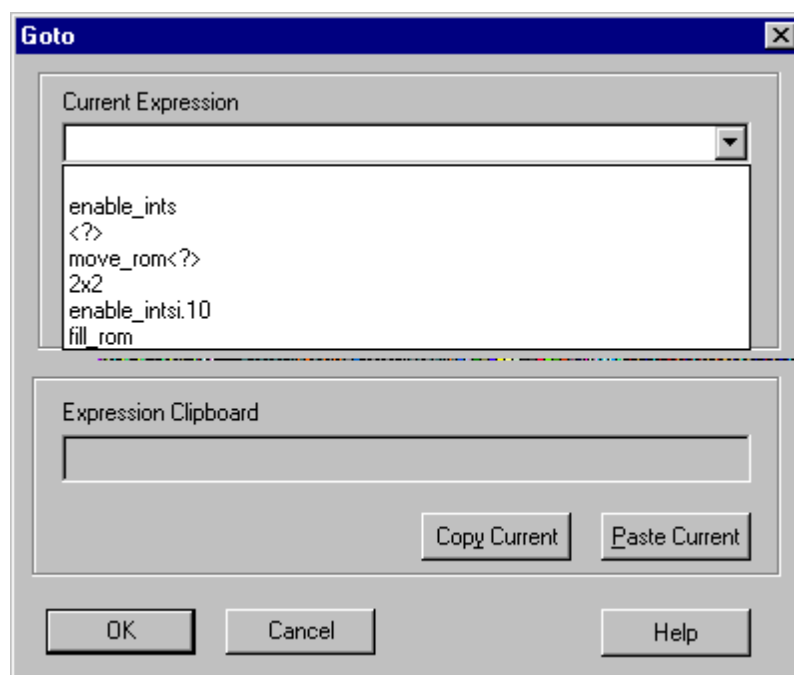


## Moving To A Known Address Or Label




The Goto command is available on the Source, Disassembly and Memory Pane menus. It is used to put the caret and PC at a known address, label name, register name or value of a specified C expression as described below:

1. Make the Source, Disassembly or Memory Pane Active.
2. Click the right hand mouse button to call the Pane menu.
3. Select the Goto option from the menu.
4. The Goto Expression dialog box appears.



*Go To Expression Dialog Box*

5. Type the required expression directly in the **Current Expression** box or click the down arrow to display expressions which have been entered previously. Various browsing and matching facilities are also available via this dialog box. **Using The Expression Manager** for further details.
6. Enter or select the required expression and click . Note that a hexadecimal address must be prefixed with the string '0x'

As the Goto action will take you to the **value** of the specified expression, note the consequences when you enter a name containing C debug information **as well** as an Assembler label.

For example, if ‘\_ramsize’ is specified you will be taken to **value** of \_ramsize, not to where it is defined. This is because the C expression evaluator sees the C definition of \_ramsize first and then evaluates it. **T**o goto this address, you must enter either ‘&\_ramsize’ or ‘:\_ramsize’.

Alternatively, you could **G**oto ‘main’ (as functions evaluate to their address); to **G**oto an offset from main, enter: ‘:main+offset’, ‘&main+offset’ or ‘(int)main+offset’. This is because ‘main’ by itself has the **typ**e ( ) which cannot be added.

**Note:** When you have successfully ‘gone to’ an expression in a Memory Pane, the ‘pointed to’ word is enclosed in a box. This will remain until you **G**oto something else or anchor the Pane to an expression.


**See Also:**

**Anchoring Memory Panes**

**Expression Evaluation Features**

**Using The Expression Manager**

**Previously Entered Expressions History List**

**Note:** Alternatively, you can activate the Goto command via the Goto icon on the toolbar  or the Hot Key **Ctrl G**.

## Expression Evaluation Features

### Register Names

Register names can be specified in any dialogue box where expressions can be entered. By default, the evaluator looks for C symbols first, so any variables which are the same as register names will be shown instead. If a name is being interpreted as a register it will be prefixed by a '\$'.

It is recommended that you use this '\$' prefix when **entering** register names to explicitly tell the evaluator that it is looking at a register.

---

**Note:** Registers have a C type of 'int'.

---

### Typecasts and Typedefs

Typecasts can be entered to an expression via the usual C syntax. If you entered '(int\*)\$fp' to a Watch Pane you would see the following:

**+int\*(int\*)\$fp = 0x8000ff00**

Typecasting also works for structure tags; however, you are not required to enter the keyword 'struct' when casting to a structure tag.

You would expect to see the following when typecasting to a structure (or class):

**-Tester\* (Tester\*)\$fp = 0x807ff88**  
**-Tester**  
**+unsigned char\* m\_pName = 0x00000645**  
**+unsigned char\* mpLongName = 0xFFFFFFFF**

You can also cast to typedefs; for example, entering '(daddr\_t)p' will produce:

**long (daddr\_t)p = 0x00003024**

## Labels

Labels can also be included in a C expression. The evaluator looks for C level information first and then label information. If it finds a label it will prefix it with a ':'.

It is recommended that you use this prefix when **entering** labels to explicitly tell the evaluator that it is looking at a label.

---

**Note:** Labels have a C type of **int**'.

---

## Functions

If you include a function name in an expression, its value will be the same as its address. It will appear in a Watch window as follows:

**int ( ) main = (...) (0x80010BFC)**

---

**Note:** This is contrary to C where the value of a function, is what is returned from the function when it is executed.

---

---

**Note:** It is recommended that you access the address of a function via the '&' operator or the Assembler label.

---

## Expression Evaluation Name Resolution


In summary, the search order for a name in an expression is as follows:

1. Escaped Register Names (prefix '\$')
2. Escaped Label Names (prefix ':')
3. C Names
4. Register Names
5. Label Names

## Previously Entered Expressions History List

Once an expression has been specified via an Expression Manager dialogue box, it will be stored in a history buffer.

When you next access one of these dialogue boxes, click the down arrow to display a listing of these values.

At this point you can enter a new expression to the dialogue box or select one from the list and click . The selected expression can also be edited at this point.

**Note:** The most recent expressions used are held at the top of the list.

## Anchoring Panes To The PC


By default the Source and Disassembly Panes are anchored to the Program Counter (PC).

This means that whenever possible the instruction or line at the PC is always displayed in the Pane.



The Follow PC property is toggled as follows

1. Make a Source or Disassembly Pane Active.
2. Click the right hand mouse button to call the Pane menu.
3. Select the Follow PC option from the menu.

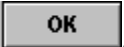
**Note:** This option is also available from the Source and Disassembly Pane toolbar  or from the Source or Disassembly menus on the Menu bar.

## Anchoring Memory Panes

Anchoring a Pane has the same function as using Goto on every Debugger update.



To anchor the Pane:


1. Select Anchor... from the Pane menu or press **Ctrl+A** when a Memory Pane is active.
2. Enter an expression.
3. Select .

The specified expression will appear in an indicator bar on the Pane. If this goes red, the expression cannot be evaluated in the current scope. Otherwise, the Pane will be 'anchored' to the value of the expression and a box will be drawn around the anchor point.

You can edit the expression by clicking the indicator.



To turn off anchoring:

1. Call up the Anchor dialogue box.
2. Clear the box.
3. Select .

## Identifying Changed Information

Any changes to variables since the last debugging step are displayed in a colour of your choice on allPanels except for Disassembly and Source.

This colour is set via the Set DefaultColour option from theView menu.

**See Also:**

Changing colour schemes in Views

## Closing The Debugger Without Saving Your Changes

The Quit option on the Project menu stops the Debugger running but does NOT save the current Project.



To close the Debugger without saving your changes:

1. Select the Project menu from the Menu bar.
2. Choose the Quit option from the menu.

**Note:** When you close the Debugger, it will remember the current working directory and restore this when it is re-opened.

## Closing The Debugger And Saving Your Changes

The **E**xit option on the Project menu saves the current Project at the latest state and stops the Debugger running.



To close the Debugger and save your changes

1. Select the **P**roject menu from the Menu bar.
2. Choose the **E**xit option from the menu.

**Note:** It is also possible to close the Project by clicking on the **X** icon on the system menu shown in the top right corner of the Debugger window.

**Note:** Next time you open the Debugger the last Project you saved will be launched automatically.

**Note:** When you close the Debugger, it will remember the current working directory and restore this when it is re-opened.

**See Also:**

**Saving Your Project**



**CHAPTER 11****The PSYLINK Linker**

The Linker **PSYLINK** is a fully-featured linker which works with all processor types. You can split complex programs into separate, manageable sub-programs which can be recombined by **PSYLINK** into a final, single application.

This chapter discusses the linker together with the Librarian utility, under the following headings:

- **Command Line Syntax**
- **Linker Command Files**
- **XDEF, XREF and PUBLIC**
- **GLOBAL**

The Linker-associated Assembler directives (XDEF, XREF, PUBLIC and GLOBAL) are repeated here for ease of reference.

## PSYLINK Command Line

**Description** The **PSYLINK** link process is controlled by a series of parameters on the command line and optionally by the contents of a Linker command file. The syntax for the command line is as follows:

**Syntax** **PSYLINK** [*switches*] *objectfile(s),output,symbolfile,mapfile,libraries*

If a parameter is omitted, the separating comma must still appear unless it is the last parameter of the line.

**Switches** Switches are preceded by a forward slash (/) and are as follows:

<b>/c</b>	Tells the linker to link case sensitive; if it is <b>omitted</b> , all names are converted to upper case.
<b>/d</b>	Debug Mode - perform link only.
<b>/e <i>symb=value</i></b>	Assigns value to symbol.
<b>/i</b>	Invokes a window containing Link details.
<b>/l <i>path</i></b>	Specify path to search for library files.
<b>/m</b>	Output all external symbols to the map file.
<b>/n <i>maximum</i></b>	Set the maximum number of object files, or library modules, that can be linked, 1 to 32768; default = 256 Higher values require larger amounts of memory.
<b>/o <i>address</i></b>	Set an address for an <b>ORG</b> statement.
<b>/p</b>	Output padded pure binary object code; <b>ORG</b> ed sections of code are separated with random data. (equivalent of <b>pb</b> switch on assembler)
<b>/ps</b>	Output ASCII representation of binary file in Motorola <i>s-record</i> format.
<b>/s</b>	All sections must be in defined groups.
<b>/u <i>number</i></b>	Specify the <i>unit number</i> in a multi-processor target.
<b>/v</b>	Enable automatic overlay recognition by the Debugger

---

<b>/wl</b>	Warn of multiple definitions in libraries.
<b>/wm</b>	Warn of multiple declarations of the same C variable.
<b>/x <i>address</i></b>	Set address for the program to commence execution.
<b>/z</b>	Clear all requested BSS memory sections.

**Objectfile(s)** A list of object files, output by the Assembler using the **tl** option. File names are separated by spaces or plus (+) signs; if the file starts with a **@** sign, it signifies the name of a Linker command file. See below for a description of the format.

**Output** The destination file for the linked code; if omitted no output code is produced. If the output file name is in the format **T:**, the linked code is directly sent to the target machine **-n** specifies the SCSI device number.

**Symbolfile** The destination file for the symbol table information used by the Debugger.

**Mapfile** The destination file for map information.

**Libraryfiles** Library files available - see The PSYLIB Librarian chapter for further information.

## Linker Command Files

Command files contain instructions for the Linker about source files and how to organise them. The Linker command file syntax is much like the Assembler syntax, with the following commands available:

<b>Commands</b>	<b>INCLUDE</b>	<i>filename</i>	Specify name of object file to be read.
	<b>INCLIB</b>	<i>filename</i>	Specify library file to use
	<b>INCBIN</b>	<i>filename</i>	Specify binary file to be included
	<b>ORG</b>	<i>address</i>	Specify <b>ORG</b> address for output
<i>name</i>	<b>EQU</b>	<i>value</i>	Equate name to value
	<b>REGS</b>	<i>pc=address</i>	Set initial <b>PC</b> value
<i>name</i>	<b>GROUP</b>	<i>attributes</i>	Declare group
<i>name</i>	<b>SECTION</b>	<i>attributes</i>	Declare section with attributes
	<b>SECTION</b>	<i>name[,group]</i>	Declare section, and optionally specify its group
<i>name</i>	<b>ALIAS</b>	<i>oldname</i>	Specify an <b>ALIAS</b> for a symbol name
	<b>UNIT</b>	<i>unitnum</i>	Specify destination unit number; this must always be 0 for the Nintendo.

### Group attributes:

<b>BSS</b>	group is uninitialised data
<b>ORG</b> ( <i>address</i> )	specify group's org address
<b>OBJ</b> ( <i>address</i> )	specify group's obj address
<b>OBJ</b> ( )	group's obj address follows on from previous group
<b>OVER</b> ( <i>group</i> )	overlay specified group
<b>FILE</b> ("filename")	write group's contents to specified file
<b>SIZE</b> ( <i>maxsize</i> )	specify maximum allowable size

### Remarks

- The directive **INCBIN** allows direct inclusion of binary data. The command format is: `label INCBIN file,section[,alignment]` e.g.

```
grdata incbin "graphics.bin",.data
```

This will put the contents of binary file `graphics.bin` into section `.data` and define label `grdata` to be at the start of this data.

The optional alignment specifies the data's alignment as a power of 2, for example in:

```
grdata incbin "graphics.bin",.data,4
```

the data will be aligned on a 16 byte boundary.

- Both INCLUDE and INCLIB statements can specify a prefix to be added to all the section names specified in a module/library, for example,

```
include "l1.obj",level1
```

will cause all sections mentioned in the file `l1.obj` to have `level1` prefixed onto them so `.text` becomes `level1.text`, etc.

This removes the need to use the previously used `Wa,s...` option in `ccn64/ccpsx/etc`.

If INCLIB has a prefix specified then all used modules from that library will have the prefix added to their section's names.

- The Linker will now define global labels to allow the user to access the base address and size of each section and group. This is done by prefixing each group and section name with `_` (underscore) and adding `_obj`, `_org` and `_size` onto the end of the name to produce three names giving the `obj`, `org` and `size` of the group/section respectively. Any `.` (dot) characters in the name will be converted to `_` (underscore) characters.

For example, the existence of the standard section `.text` will result in the creation of the labels `__text_obj`, `__text_org` and `__text_size`.

**Note:** Two leading underscores, one explicitly added and one from converting the `'.'`.

To access these names from C, declare the name as an external and then take its address to get the value, e.g.

```
memset(&__bss_obj, 0, (int) &__bss_size);
```

will clear the `bss` section to 0.

- Sections within a group are in the order that section definitions are encountered in the command file or object/library files.
- Any sections that are not placed in a specified group will be grouped together at the beginning of the output.
- Groups are output in the order in which they are declared in the Linker command file or the order in which they are encountered in the object and library files.
- Sections which are declared with attributes, (i.e. not in a group) in either the object or library files, may be put into a specified group by the appropriate declaration in the Linker command file.

<b>Examples</b>		include	"inp.obj"
		include	"sort.obj "
		include	"out.obj"
		org	1024
		regs	pc=progstart
	comdata	group	word
	code	group	
	bssdata	group	bss
		section	data1,comdata
		section	data2,comdata
		section	code1,code
		section	code2,code
		section	tables,bssdata
	section	buffers,bssdata	

---

## Linker Associated Directives.

### GLOBAL

**Description** The **GLOBAL** directive allows a symbol to be defined which will be treated as either an **XDEF** or an **XREF**. If a symbol is defined a **GLOBAL** and is later defined as a label, it will be treated as an **XDEF**. If the symbol is never defined, it will be treated as an **XREF**.

**Syntax**                      **GLOBAL**    *symbol[,symbol]*

**See Also**            **XREF, XDEF, PUBLIC**

**Remarks**            This is useful in header files because it allows all separately assembled modules to share one header file, defining all global symbols. Any of these symbols later defined in a module will be **XDEF**ed, the others will be treated as **XREF**s.

## XDEF, XREF and PUBLIC

**Description** If several sub-programs are being linked, use XDEF, XREF and PUBLIC to refer to symbols in a sub-program which are defined in another sub-program.

**Syntax**

<b>XDEF</b>	<i>symbol</i> [, <i>symbol</i> ]
<b>XREF</b>	<i>symbol</i> [, <i>symbol</i> ]
<b>PUBLIC</b>	<b>on</b>
<b>PUBLIC</b>	<b>off</b>

### Remarks

- In the sub-program where symbols are initially defined, the **XDEF** directive is used to declare them as externals.
- In the sub-program which refers the symbols, the **XREF** directive is used to indicate that the symbols are in another sub-program.
- The Assembler does not completely evaluate an expression containing **XREF**ed symbol; however, resolution will be effected by the linker.
- The **PUBLIC** directive allows the programmer to declare a number of symbols as externals. With a parameter of on, it tells the Assembler that all further symbols should be automatically **XDEF**ed, until a **PUBLIC** off is encountered.

**Examples** Sub-program A contains the following declarations :

```
xdef      Scores , Scorers
...
```

The corresponding declarations in sub-program B are:

```
xdef      PointsTable
xref      Scores , Scorers
...

Origin    =
Force     dcw      speed*origin
Rebound   dcw      45*angle
public    off
```

**Example Linker Command File for GNU C Program**

```
org                $80010000
text               group
bss                group      bss

section            .rdata, text
section            .text, text
section            .ctors, text
section            .dtors, text
section            .data, text
section            .sdata, text
section            .sbss, bss

include            objfile1.obj
include            objfile2.obj

inclib             c:\n64\lib\libs.n.lib

regs               PC=__SN_ENTRY_POINT
```



**CHAPTER 12****The PSYLIB Librarian**

If the Linker cannot find a symbol in the files produced by the Assembler, it can be instructed by a Linker command line option to search one or more object module Library files.

This chapter discusses Library usage and the **PSYLIB** library maintenance program:

- **PSYLIB Command Line Syntax**
- **Using the Library feature**

## PSYLIB Command Line

**Description** The Library program, **PSYLIB.EXE**, adds to, deletes from, lists and updates libraries of object modules.

**Syntax** **PSYLIB** [*switches*] *librarymodule...module*

where a switch is preceded by a forward slash (/)

**See Also** **PSYLINK**

**Switches**

- /a** Add the specified module(s) to the library, replacing any already present
- /d** Delete the specified module(s) from the library
- /l** List the module(s) contained in the library (all will be listed if not specified)
- /lv** List specified library modules including symbols, if changed
- /u** Update the specified modules in the library
- /x** Extract the specified modules from the library (extracts all modules if no module list given)

**Note:** The **/lv** switch shows symbols in the order they're declared in the modules. XBSS-type symbols appear with an asterisk before the name, e.g. **\*fbuff**.

**Library** The name of the file to contain the object module library.

**Module list** The object modules involved in the library maintenance.

### Using the Library feature

- To incorporate a Library at link time, specify a library file on the Linker command line - see chapter 12.
- If the Linker locates the required external symbol in a nominated library file, the module is extracted and linked with the object code output by the Assembler.

## CHAPTER 13

## The CCN64 Build Utility

**CCN64** is a *build* utility to execute the C Compiler, Assembler and Linker. **CCN64** makes use of the **ASN64** Assembler to process the assembly syntax produced by the GNU C Compiler. It is discussed in the following sections:

- **CCN64 Command Line**
- **Source Files**

## CCN64 Command Line

**Description** **CCN64.EXE** is a utility that simplifies the process of running the separate 'C' compiler passes and then assembling and linking the compiler output. Using **CCN64** you need only specify the input files and the output format you require, and **CCN64** itself will execute the tools required to generate the output.

**Syntax** **CCN64** [*options* / *filename* [, *filename*...]]

The command line consists of a sequence of control options and source file names. *Options* are preceded by a minus sign (-), and *filenames* are separated by commas.

Long command lines can be stored in separate control files. These can then be used on the command line by using a '@' sign in front of the control file name.

### Options

#### *Control*

- E** Pre-process only. If no output file is specified output is sent to the screen.
- S** Compile to assembler source
- c** Compile to object file. If an output file is specified, then all output is sent to this file. Otherwise it is saved as the source file name with an .OBJ extension.
- Ipath** Specify extra include path for pre-compiler.

#### *Debug*

- g...** Generate debug information for source level debugging

#### *Optimisation*

- O0** No optimisation (default)
- O or -O1** Standard level of optimisation
- O2** Full optimisation
- O3** Full optimisation and function in-lining.

#### *General*

- W...** Suppress all warnings
- Dname=val** Define pre-processor symbol *name*, and optionally to the value specified.
- Uname** Undefine the pre-defined symbol *name* before pre-processing starts
- v** Print all commands before execution
- f...** Compiler option
- m...** Machine specific option

*Linker*

- llibname** Include specified library *libname* when linking
- X...** Specify linker option
- o *destin*** Specify the destination. Either a file (e.g. prog.cpe), or target (e.g. t0:) can be specified

*See GNU C documentation for full description*

**Example**   CCPSX -v -g -Xo\$80010000 main.c -omain.cpe,main.sym

This example will execute the compiler to compile the source file **MAIN.C**, then run **ASN64** to produce the object file and finally will run **PSYLINK** to produce an executable and symbol file **MAIN.CPE** and **MAIN.SYM** respectively) **ORGd** to the specified address. The **-v** switch will cause **CCN64** to echo all commands it executes to stdout. The **-g** switch will request full debug info in the symbol file.

```
CCN64 @main.cf -omain
```

This will force **CCN64** to use the contents of the **MAIN.CF** file on the command line, before the **-o** option.

## Source Files

The specified source files can be either C or assembler source files, or object files. CCN64 decides how to deal with a source file based on the files extension. The following table describes how each file extension is processed:

<b>.c</b>	Passed through C pre-processor, C compiler, Assembler, Linker
<b>.i</b>	Passed through C compiler, Assembler, Linker
<b>.cc</b>	Passed through C pre-processor, C++ compiler, Assembler, Linker
<b>.cpp</b>	Passed through C pre-processor, C++ compiler, Assembler, Linker
<b>.ii</b>	Passed through C++ compiler, Assembler, Linker
<b>.ipp</b>	Passed through C++ compiler, Assembler, Linker
<b>.asm</b>	Passed through C compiler, Assembler, Linker
<b>.s</b>	Passed through Assembler, Linker
<b>.other</b>	Passed through Linker

### Remarks

- The PC file system is not case sensitive so the case of the extension has no effect. The UNIX file system is case sensitive so the case of the extension is important.
- Various command line switches can stop processing at any stage, eliminating linking, assembling or compiling.
- The -x option can be used to override the automatic selection of action based on file extension.
- Files with extensions that are not recognised are treated as object files and passed to the linker. This includes **OBJ** files, the standard object file extension.
- Several different source files, which may have different file extensions, may be placed on the command line.

**CHAPTER 14****The PSYMAKE Utility**

**PSYMAKE** is a make utility which automates the building and rebuilding of computer programs. It is general purpose and not limited to use with this system. The utility is discussed under the following headings:

- **Command Line Syntax**
- **Format of the Makefile**

## PSYMAKE Command Line

**Description** PSYMAKE only rebuilds the components of a system that need rebuilding. Whether a program needs rebuilding is determined by the file date stamps of the target file and the source files that it depends on. Generally, if any of the source files are newer than the target file, the target file will be rebuilt.

**Syntax** PSYMAKE [*switches*] [*target file*]

or

PSYMAKE @*makefile.mak*

**Switches** Valid switch options are :

<b>/b</b>	Build all, ignoring dates
<b>/d</b> <i>name=string</i>	Define name as string
<b>/f</b> <i>filename</i>	Specify the <b>MAKE</b> file
<b>/i</b>	Always ignore error status
<b>/q</b>	Quiet mode; do not print commands before executing them
<b>/x</b>	Do not execute commands - just print them

If no **/f** option is specified, the default makefile **makefile**. If no extension is specified on the makefile name **MAK** will be assumed.

If no target is specified, the first target defined in the makefile will be built.



## Contents of the Makefile

The Makefile consists of a series of commands, governed by explicit rules, known as dependencies, and implicit rules. When a target file needs to be built, **PSYMAKE** will first search for a dependency rule for that specific file. If none can be found, **PSYMAKE** will use an implicit rule to build the target file.

### Dependencies:

A dependency is constructed as follows :

```
targetfile : [ sourcefiles ]
               [ command
               ...
               command ]
```

- The first line instructs **PSYMAKE** that the file "*targetfile*" depends on the files listed as "*sourcefiles*".
- If any of the source files are dated later than the target file, or the target file does not exist, **PSYMAKE** will issue the commands that follow in order to rebuild the target file.
- If no source files are specified, the target file will always be rebuilt.
- If any of the source files do not exist, **PSYMAKE** will attempt to build them first before issuing the commands to build the current target file. **PSYMAKE** cannot find any rules defining how to build a required file, it will stop and report an error.

The target file name must start in the left hand column. The commands to be executed in order to build the target must all be preceded by white space (either space or tab characters). The list of commands ends at the next line encountered with a character in the leftmost column.

**Examples**    `main.cpe: main.n64 incl.h inc2.h  
                  asmn64 main,main`

This tells **PSYMAKE** that `main.cpe` depends on the files `main.n64`, `incl.h` and `inc2.h`. If any of these files are dated later than `main.cpe`, or `main.cpe` does not exist, the command "`asmn64 main,main`" will be executed in order to create or update `main.cpe`.

```
main.cpe: main.n64 incl.h inc2.h
          asmn64 /l main,main,main
          psylink main,main
```

Here, two commands are required in order to rebuild `main.cpe`.

### Implicit Rules

If no commands are specified **PSYMAKE** will search for an implicit rule to determine how to build the target file. An implicit rule is a general rule stating how to derive files of one type from another type; for instance, how to convert `asm` files into `.exe` files.

Implicit rules take the form:

```
.<source extension>.<target extension>:  
  command  
  [...  
  command ]
```

Each `<extension>` is a 1, 2 or 3 character sequence specifying the DOS file extension for a particular class of files.

At least one command must be specified.

**Examples**    `.s.bin:  
                  asmn64 /p $*,$*`

This states that to create a file of type `bin` from a file of type `S`, the `asmn64` command should be executed. (See below for an explanation of the `$*` substitutions.)

### Executing commands

Once the commands to execute have been determined, **PSYMAKE** will search for and invoke the command. Search order is:

- current directory;
- directories in the path.

### Command prefixes

The commands in a dependency or implicit rule command list, may optionally be prefixed with the following qualifiers :

- @ - suppress printing of command before execution
- *number* - abort if exit status exceeds specified *number*
- - (without number) ignore exit status (never abort)

- Normally, unless **q** is specified on the command line, **PSYMAKE** will print a command before executing it. If the command is prefixed by @, it will not be printed.
- If a command is prefixed with a hyphen followed by a number, **PSYMAKE** will abort if the command returns an error code greater than the specified number.
- If a command is prefixed with a hyphen without a number, **PSYMAKE** will not abort if the command returns an error code.
- If neither a hyphen or a hyphen+number is specified, and **q** is not specified on the command line, **PSYMAKE** will abort if the command returns an error code other than 0.

**Macros** A macro is a symbolic name which is equated to a piece of text. A reference to that name can then be made and will be expanded to the assigned text. Macros take the form:

```
name = text
```

- The *text* of the macro starts at the first non-blank character after the equals sign (=), and ends at the end of the line.
- Case is significant in macro names.
- Macro names may be redefined at any point.
- If a macro definition refers to another macro, expansion takes place at time of usage.
- A macro used in a rule is expanded immediately.

**Examples**

```
FLAGS = /p /s
...
.s.bin:
    asmn64 $(FLAGS) /p $*,$*
```

The  $\$(FLAGS)$  in the **asmn64** command will be replaced with **p/s**.

### Pre-defined macros

The following pre-defined macros all begin with a dollar sign and are intended to aid file usage:

<b>\$d</b>	Defined Test Macro, e.g.:	
	!if \$d(MODEL)	
	# if MODEL is defined ...	
<b>\$*</b>	Base file name with path,	e.g. C:\PSYQ\TEST
<b>\$&lt;</b>	full file name with path,	e.g. C:\PSYQ\TEST.S
<b>\$:</b>	path only,	e.g. C:\PSYQ
<b>\$.</b>	full file name, no path,	e.g. TEST.s
<b>\$&amp;</b>	base file name, no path,	e.g. TEST

The filename pre-defined macros can only be used in command lists of dependency and implicit rules.

**Directives:** The following directives are available:

```
!if    expression
!elseif expression
!else
!endif
```

These directives allow conditional processing of the text between *!if*, *!elseif*, *!else* and *!endif*. Any non-zero expression is *true*; zero is *false*.

```
!error message    Print the message and stop.
```

```
!undef macro name  Undefines a macro name.
```

**Expressions** Expressions are evaluated to 32 bits, and consist of the following components :

```
Decimal Constants  e.g. 1 10 1234
Hexadecimal       e.g. $FF00 $123abc
Monadics          - ~ !
Dyadics           + - * / % > < &
                    | ^ && ||
                    > < >= <= == (or = )
                    != (or <> )
```

The operators have the same meanings as they do in the C language, except for = and <>, which have been added for convenience.

### Value assignment

Macro names can be assigned a calculated value; for instance:

```
NUMFILES == $(NUMFILES)+1
```

( Note **two** equals signs in value assignment)

This evaluates the right hand side, converts it to a decimal ascii string and assigns the result to the name on the left.

In the above example, if *NUMFILES* was currently "42", it will now be "43".

### Note

```
NUMFILE = $(NUMFILES)+1
```

would have resulted in *NUMFILES* becoming "42+1".

Undefined macro names convert to '0' in expressions and null string elsewhere.

**Comments:**

Comments are introduced by a hash mark (#)

```
main.exe: main.n64    # main.exe only depends
                   # on main.n64
```

```
# whole line comment
```

**Line continuation:**

A command too long to fit on one line may be continued on the next by making '\' the last character on the line, with no following spaces/tabs:

```
main.exe : main.n64 i1.h i2.h \  
          i3.h i4.h
```

## TBIOS2.COM

**Description:** TBIOS2.COM is a DOSTSR program, which acts as a driver for the interface board installed in the host PC

**Note:** This program is only necessary if you are running DOS software.

**Syntax**      TBIOS2      [*switches*]

where each switch is preceded by a forward slash(/) and separated by spaces.

<b>Switches</b>	<i>/a</i>	<i>card address</i>	Set card address: <b>200 - 3f8</b>
	<i>/b</i>	<i>buffer size</i>	Specify file transfer buffer size <b>2 to 32</b> (in kilobytes)
	<i>/d</i>	<i>channel</i>	Specify DMA channel <b>5, 6, 7; 0 = off</b>
	<i>/i</i>	<i>intnum</i>	Specify IRQ number <b>5, 7, 10, 11, 12, 15; 0 = off</b>
	<i>/l</i>	<i>filename</i>	Specify Fileserver log file; All fileserver functions will be recorded in the specified file.
	<i>/s</i>	<i>id</i>	Override PC Interface SCSI ID jumper setting: <b>0 to 7</b>
	<i>/8</i>		Run in 8 bit slot mode

### Remarks

- Normally, TBIOS2.COM is loaded in the AUTOEXEC.BAT. It can safely be loaded high to free conventional memory.
- If TBIOS2 is run again, with no options, the current image will be removed from memory. This is useful if you wish to change the options without rebooting the PC.
- If the DMA number is not specified, the TBIOS2 will work without DMA; however, it will be slower.
- The TBIOS2 can drive the interface in 8 bit mode; however, this is the slowest mode of operating the interface.
- The *buffer size* option (/b) sets the size of the buffer used when the target machine accesses files on PC. A larger buffer will increase the speed of these accesses; however, more PC memory will be consumed. The default is a 1 K buffer.

**Examples**    TBIOS2 /a308 /d7 /i15

Start the driver using the typical settings of:

Card address <b>308</b> DMA channel <b>7</b> Interrupt vector <b>15</b>
---

**Note:** The downloaded bios will remain until the interface is powered off. It should not be necessary to switch it off during the normal course of development. It is possible to switch the N64 off and on or reset it without affecting the bios in the interface. The only thing that can cause the interface to lock up is if the Nintendo 64 saturates access to the cartridge ram. If this happens switch the Nintendo off and allow it to recover.

**Note:** If you are using Windows 95, DO NOT install TBIOS2 from your AUTOEXEC.BAT. Only install it in one DOS box.



## RUN.EXE - program downloader

**Description** This program downloads runnable object code to the target machine.

**Note:** TBIOS2 must be loaded in the current DOS box.

**Syntax** RUN [*switches*] *file name* [[*switches*] *filename...*]

where switches are preceded by a forward slash (/)

**Switches**

- /h** halt target - that is, download but do not run.
- /t#** use target SCSI IDnumber# - *always 0 for the Nintendo.*
- /u#** use target unit number# *always 0 for the Nintendo*
- /w##** retry for ## seconds if target does not respond.

**Remarks** If run is executed without any runtime parameters, the program will simply attempt to communicate with the target adapter hardware. If successful, run displays the target identification; if the attempt fails, an appropriate error message is displayed.

The file to be downloaded may contain:

- An executable image, output by the development system, in .cpe format. Up to 8 cpe files may be specified
- A raw binary image of a cartridge.

For an executable file, execution will begin as indicated in the source code; for a binary ROM image, execution will begin as if the target machine had been reset with a cartridge in place.

Multiple executable files may be specified. However, only the last executable address will apply - specified files are read from left to right.

A binary file can be downloaded to a particular address by specifying the address after the file name, e.g.:

```
run game.bin,b0000000
```

will download game.bin to address b0000000 (hex).

**Note:** The Windows equivalent to this program is `isqrun`.

## Running with Brief

Most programmers prefer to develop programs completely within a single, enabling environment. Future versions of the software will provide a self-contained superstructure with a built-in editor, tailored to the requirements of the assembly and debug sub-systems. For the time being however, these facilities can be found in Borland's Brief text editor.

### Installation in Brief

In order to use Brief you will need to make a few changes to your AUTOEXEC.BAT file after you have installed Brief:

Set the BCxxx environment variables. These variables take the file extension of a source file to tell Brief how to Assemble or Compile the file. For example:

```
set bcn64="asmn64 /i /w /d /zd %s,t0:,%s,,"
set bcs="asmn64 /i /w /d /zd %s,t0:,%s,,"
set bcc="ccn64 -v -g -Xo$80010000 %s.c -
o%s.cpe,%s.sym"
```

These will Assemble .N64 source file with ASMN64, .S source files with the ASMN64 assembler (see chapter 2), and Compile .C source files with CCN64 (see The Build Utility chapter).

Set the BFLAGS environment variable, with -mPSYQ appended, to force the macro file to be loaded on start-up. For example:

```
set bflags=-ai60Mk2u300p -mrestore -Dega -D101key -mPSQ
```

The variable may look different depending on how Brief was set up.

Finally, copy the file PSYQ.CM, containing macros, into the \BRIEF\MACROS directory, or create it from source file PSYQ.CB;

**Note:** The standard Brief feature of using **Alt-F10** to compile the current file as instructed by the **BCxx** environment variable still works as normal. However, if you take the time to write a simple make file for each of your projects you will find the System's keystrokes much more convenient and powerful.

### The brief macros

**Ctrl-G** Goto label (locate definition of label under the cursor in loaded source files)  
**Ctrl-F** Return from label (undo the above operation)  
**Ctrl-W** Write out all changed files  
**Ctrl-V** Evaluate expression under cursor using values from symbol file(s)  
**Ctrl-F9** Select make file for current project  
**Ctrl-F10** Make program and enter debugger  
**Alt-F9** Make program and start it running  
**F9** Enter the Debugger

If you wish to change any of these key assignments then change to your `\BRIEF\MACROS` directory and edit the file `PSYQ.CB`. Near the top of this file you will see where the keys are assigned to the various functions and it should be easy to change the key names and re-compile the macro by pressing **Alt-F10**.

**Note:** If you re-assign any of the Brief standard key assignments then you may lose access to that original Brief function.

**Ctrl-F9** allows you to select which make file you wish to use. By default, the System will use the file in your current directory called **MAKEFILE.MAK**. If you do not wish to use this file then use **Ctrl-F9** to select the preferred make file.

**Ctrl-F10**, **Alt-F9** and **F9** work by calling the `PSYMAKE` program with a suitable parameter to select which operation to perform. Your System disk includes a simple make file called `MAKEFILE.MAK` as an example. If you edit this file you will see that it defines how to do one of three operations:-

- Assemble and Run
- Assemble but don't Run
- Enter Debugger

It should be easy for you to adapt this file to your needs. If you are doing one simple assembly then all you will have to change is the name of the file that is assembled and add any other command line options you require.

It does not matter which of your source files you are in when you press one of the make/debug keys - the make file will specify the commands to the assembler and debugger.





## Assembler Error Messages

### Assembler Messages:

**'%n' cannot be used in an expression**

%n will be the name of something like a macro or register

**'%n' is not a group**

Group name required

**'%n' is not a section**

Section name expected but name %n was found

**Alignment cannot be guaranteed**

Warning of attempt to align that cannot be guaranteed due to the base alignment of the current section

**Alignment's parameter must be a defined name**

In call to alignment( ) function

**Assembly failed**

Text of the FAIL statement

**Bad size on opcode**

E.g. attempt to use .b when only .w is allowed

**Branch (%l bytes) is out of range**

Branch too far

**Branch to odd address**

Warning of branch to an odd address

**Cannot POPP to a local label**

E.g. POPP @x

**Cannot purge - name was never defined**

**Case choice expression cannot be evaluated**

On case statement

**Code generated before first section directive**

Code generating statements appeared before first section directive

**Could not evaluate XDEF'd symbol**

XDEF'd symbol was equated to something that could not be evaluated

**Could not open file '%s'**

**Datasize has not been specified**

Must have a DATASIZE before DATA statement

**Datasize value must be in range 1 to 256**

DATASIZE statement

**Decimal number illegal in this radix**

Specified decimal digit not legal in current radix

**DEF's parameter must be a name**

Error in DEF( ) function reference

**Division by zero**

**End of file reached without completion of %s construct**

E.g. REPT with no ENDR

**ENDM is illegal outside a macro definition**

**Error closing file**

System close file call returned an error status

**Error creating output file**

Could not open the output file

**Error creating temporary file**

Could not create specified temporary file

**Error in assembler options**

**Error in expression**

Similar to syntax error

**Error in floating point number**

In IEEE32 / IEEE64 statement

**Error in register list**

Error in specification of register list for MOVEM / REG

**Error opening list file**

System open returned an error status

**Error reading file**

System read call returned an error status



**Error writing list file**

An error occurred while writing to the list file.

**Error writing object file**

System write call returned an error or disk is full

**Error writing temporary file**

Disk write error, probably disk full

**Errors during pass 1 - pass 2 aborted**

If pass 1 has errors then pass 2 is not performed

**Expanded input line too long**

After string equate replacement, etc. line must be <= 1024 chars

**Expected comma after < >**

<...> bracketed parameter in MACRO call parameter list

**Expected comma after operand****Expected comma between operands****Expected comma between options**

In an OPT statement

**Expecting '%s' at this point**

Expecting one of ENDIF/ENDCASE etc. but found another directive

**Expecting '+' or '-' after list command**

In a LIST statement

**Expecting '+' or '-' after option**

In an OPT statement

**Expecting a number after /b option**

On Command line

**Expecting comma between operands in INSTR****Expecting comma between operands in SUBSTR****Expecting comma or end of line after list**

In { ... } list

**Expecting ON or OFF after directive**

In PUBLIC statement

**Expecting options after /O**

On Command line

**Expecting quoted string as operand****Expression must evaluate**

Must be evaluated now, not on pass 2

**Fatal error - macro exited with unterminated %s loop**

End of macro with unterminated WHILE/REPT/DO loop.

Due to the way the assembler works, this must be treated as a fatal error

**Fatal error - stack underflow - PANIC**

Assembler internal error - should never occur!

**File name must be quoted****Files may only be specified when producing CPE or pure binary output**

In FILE attribute of group

**Forward reference to redefinable symbol**

Warning that a forward reference was made to a symbol that was given a number of values in SET or = statements. The value used in the forward reference was the last value the symbol was set to.

**Function only available when using sections****Group '%n' is too large (%l bytes)**

Group exceeds value in SIZE attribute

**GROUP's parameter must be a defined name**

In GROUP( ) function call

**GROUPEND's parameter must be a group name**

Error in call to GROUPEND( ) function

**GROUPORG's parameter must be a group**

In call to GROUPORG( ) function

**GROUPSIZE's parameter must be a group name**

Error in call to GROUPSIZE( ) function

**IF does not have matching ENDIF/ENDC****Illegal addressing mode**

Addressing mode not allowed for current op code

- 
- Illegal character '%c' (%d) in input**  
Strange (e.g. control) character in input file
- Illegal character '%c' in opcode field**
- Illegal digit in suffixed binary number**  
In alternate number form 101b
- Illegal digit in suffixed decimal number**  
In alternate number form 123d
- Illegal digit in suffixed hexadecimal number**  
In alternate number form 1abh
- Illegal group name**
- Illegal index value in SUBSTR**
- Illegal label**  
Label in left hand column starts with illegal character
- Illegal name for macro parameter**  
In macro definition
- Illegal name in command**  
Target name in ALIAS statement
- Illegal name in locals list**  
In LOCAL statement
- Illegal name in XDEF/XREF list**
- Illegal parameter number**  
Maximum of 32 parameters
- Illegal section name**
- Illegal size specifier for absolute address**  
Can only use .w and .l on absolute addressing
- Illegal start position/length in INCBIN**
- Illegal use of register equate**  
E.g. using a register equate in an expression
- Illegal value (%1)**
- Illegal value (%1) for boundary in CNOP**
-

- Illegal value (%1) for offset in CNOP**
- Illegal value for base in INSTR**
- Initialised data in BSS section**  
BSS sections must be uninitialised
- Instruction moved to even address**  
Warning that a padding byte was inserted
- Label '%n' multiply defined**
- LOCAL can only be used inside a macro**  
LOCAL statement found outside macro
- Local labels may not be strings**  
@x EQUUS ... is illegal
- Local symbols cannot be XDEF'd/XREF'd**
- MEXIT illegal outside of macros**
- Missing '(' in function call**
- Missing ')' after function parameter(s)**
- Missing ')' after file name**  
In FILE attribute
- Missing closing bracket in expression**
- Missing comma in list of case options**  
In =... case selector
- Missing comma in XDEF/XREF list**
- MODULE has no corresponding MODEND**
- Module may not end until macro/loop expansion is complete**  
If a loop / macro call starts inside a module then there must not be a MODEND until the loop / macro call finishes
- Module must end before end of macro/loop expansion - MODEND inserted**  
A module started inside a loop / macro call must end before the loop / macro call does
- More than one label specified**  
Only one label per line (can occur when second label does not start in left column but ends in ':')

**Move workspace command can only be used when downloading**  
In WORKSPACE statement

**Names declared with local must not start with '%c'**  
In LOCAL statement

**NARG can only be used inside a macro**  
Use of NARG outside macro

**NARG's parameter must be a number or a macro parameter name**  
Illegal operand for NARG( ) function

**No closing quote on string**

**No corresponding IF**  
ENDIF/ELSE without IF

**No corresponding DO**  
UNTIL without DO

**No corresponding REPT**  
ENDR without REPT

**No corresponding WHILE**  
ENDW without WHILE

**No matching CASE statement for ENDCASE**  
ENDCASE without CASE

**No source file specified**  
No source file on command line

**Non-binary character following %**

**Non-hexadecimal character '%c' encountered**  
In HEX statement

**Non-hexadecimal character starting number**  
Expecting 0-9 or A-F after \$

**Non-numeric value in DATA statement**

**OBJ cannot be specified when producing linkable output**  
OBJ attribute on group

**Odd number of nibbles specified**  
In HEX statement

**OFFSET's parameter must be a defined name**  
Error in OFFSET( ) function call

**Old version of %n cannot be purged**

Only macros can be purged

**One string equate can only be equated to another**

Attempt to equate to expression, etc.

**Only one of /p and /l may be specified**

On Command line

**Only one ORG may be specified before SECTION directive**

**Op-code not recognised**

**Option stack is empty**

POPO without PUSHO

**Options /l and /p not available when downloading to target**

On Command line

**ORG ? can only be used when downloading output**

**ORG address cannot be specified when producing linkable output**

No ORG group attributes when producing linkable output

**ORG cannot be used after SECTION directive**

**ORG cannot be used when producing linkable output**

**ORG must be specified before first section directive**

When using sections only one ORG statement may appear before all section statements (other than as group attributes)

**Out of memory, Assembler aborting**

**Out of stack space, possibly due to recursive equates**

Assemblers stack is full, possible cause is recursive equates, e.g. x equ y+1 , y equ x\*2

**Overflow in DATA value**

DATA value too big

**Overlay cannot be specified when producing linkable output**

No OVER group attributes when producing linkable output

**Overlay must specify a previously defined group name**

Error in OVER group attribute

**Parameter stack is empty**

POPP encountered but nothing to pop

**POPP must specify a string or undefined name****Possible infinite loop in string substitution**

E.g. reference to x where x is defined as x equs x+1

**Previous group was not OBJ'd**

OBJ( ) attribute specified but previous group had no obj attribute to follow on from

**Psy-Q needs DOD version 3.1 or later.****Purge must specify a macro name****Radix must be in range 2 to 16****REF's parameter must be a name**

Error in REF( ) function reference

**Register not recognised**

Expecting a register name but did not recognise

**Remainder by zero**

As for division by 0 but for % (remainder)

**Repeat count must not be negative**

REPT statement error

**Replicated text too big**

Text being replicated in a loop must be buffered in memory but this loop was too big to fit

**Resident SCSI drivers not present.****SCSI card not present - assembly aborted**

SECT's parameter must be a defined name

Error in SECT( ) function call

**SECTEND's parameter must be a section name**

Error in call to SECTEND( ) function

**Section stack is empty**

POPS without PUSHES

**Section was previously in a different group**

Section assigned to a different group on second invocation

**SECTSIZE's parameter must be a section name**

Error in call to SECTSIZE( ) function

**Seek in output file failed**

DOS seek call returned error status

**Severity value must be in range 0 to 3**

In INFORM statement

**SHIFT can only be used inside a macro**

SHIFT statement outside macro

**Short macro calls in loops/macros must be defined before loop/macro**

**Short macros may not contain labels**

**Size cannot be specified when producing linkable output**

SIZE attribute on group

**Size specified in /b option must be in range 2 to 64**

On command line

**Square root of negative number**

**Statement must have a label**

No label on, for example, EQU op

**STRCMP requires constant strings as parameters**

**String '%n' cannot be shifted**

String specified in SHIFT statement is not a multi-element string (i.e. {...} bracketed) and so cannot be shifted.

**STRLEN's operand must be a quoted string**

**Symbol '%n' cannot be XDEF'd/XREF'd**

**Symbol '%n' is already XDEF'd/XREF'd**



**Symbol '%n' not defined in this module**  
Undefined name encountered

**Syntax error in expression**

**Timed out sending data to target**  
Target did not respond

**Too many characters in character constant**  
Character constants can be from 1 to 4 characters

**Too many different sections**  
There is a maximum of 256 sections

**Too many file names specified**  
On command line

**Too many INCLUDE files**  
Limit of 512 INCLUDE files

**Too many INCLUDE paths specified**  
Too many INCLUDE paths in /j options on command line

**Too many output files specified**  
Maximum of 256 output files

**Too many parameters in macro call**  
Maximum number of parameters (32) exceeded

**Too much temporary data**  
Assembler limit of 16m bytes of temporary data reached

**TYPE's parameter must be a name**  
Call of TYPE() function

**Unable to open command file**  
From Command line

**Undefined name in command**  
Target name in ALIAS statement

**Unexpected case option outside CASE statement**  
Found =... statement outside CASE/ENDCASE block

**Unexpected characters at end of Command line**

**Unexpected characters at end of line**

End of line expected but there were more characters encountered (other than comments)

**Unexpected end of line****Unexpected end of line in macro parameter****Unexpected end of line in list parameter**

In { ... } list

**Unexpected MODEND encountered**

MODEND without preceding MODULE

**UNIT can only be specified once**

In UNIT statement

**UNIT cannot be used when producing linkable output**

In UNIT statement

**Unknown option**

In OPT statement

**Unknown option /%c**

Unknown option on Command line

**Unrecognised attribute in GROUP directive****Unrecognised optimisation switch '%c'**

In OPT statement or Command line

**User pressed Break/Ctrl-C**

Assembly aborted by user

**XDEF'd symbol %n not defined**

Symbol was XDEF'd but never defined

**XDEF/XREF can only be used when producing linkable output****Zero length INCBIN - Warning of zero length INCBIN statement**

## Psylink Error Messages

### Linker Messages:

**%t %n redefined as section**

New definition of previously defined symbol

**%t '%n' redefined as group**

New definition of previously defined symbol

**%t '%n' redefined as XDEF symbol**

New definition of previously defined symbol

**Attempt to switch section to %t '%n'**

Non-section type symbol referenced in section switch

**Attempt to use %t '%n' as a section in expression**

Section type symbol required

**Code in BSS section '%n'**

BSS type sections should not contain initialised data

**COFF file has incorrect format**

COFF format files are those produced by Sierra C cross compiler, etc.

**Different processor type specified**

Object code is for different processor type than target or attempt was made to link code for different processor types

**Division by zero**

**Error closing file**

Close file call returned error status

**Error in /e option**

On Command line

**Error in /o option**

On Command line

**Error in /x option**

On Command line

**Error in command file****Error in Linker options**

On Command line

**Error in REGS expression****Error reading file %f**

Read file call returned error status

**Error writing object file**

Write file call returned error status - probably disk full

**Errors during pass 1 - pass 2 aborted**

Pass 2 does not take place if there were errors on Pass 1

**Expecting a decimal or hex number**

/o option on Command line

**File %f is in out-of-date format**

File should be re-built before re-assembling

**File %f is not a valid library file****File %f is not in PsyLink file format****Group '%n' is too large (%l bytes)**

Group is larger than its size attribute allows

Group '%n' specified with different attributes

Different definitions of a group specify different attributes

**Illegal XREF reference to %t '%n'**

Object file defines xref to symbol which cannot be xref'd, e.g. a Section name

**Multiple run addresses specified**

More than one run address specified

**No source files specified**

No source file on Command line

**Object file made with out-of-date assembler**

File should be re-built before re-assembling

**Only built in groups can be used when making relocatable output**

When /r command line option is used, only the built in groups can be used, i.e. no new groups may be defined

**Option /p not available when downloading to target****Options /p and /r cannot be used together**

On Command line

**ORG ? can only be used when downloading output****Out of memory, Linker aborting****Previous group was not OBJ'd**

Cannot specify OBJ() attribute if previous group did not have obj attribute

**Reference to %t '%n' in expression**

Use of, e.g. a section name in an expression

**Reference to undefined symbol %#h**

There is an internal error in the object file

**Relocatable output cannot be ORG'd****Remainder by zero****Run-time patch to odd address**

Warning that a run-time longword patch to an odd address will occur which may cause some Amiga systems to crash

**SCSI Card not present - Linking aborted**

Could not find SCSI Card

**SCSI drivers not loaded****Section '%n' must be in one of groups code, data or BSS**

When producing Amiga format code

**Section '%n' placed in non-group symbol #h**

There is an internal error in the object file

**Section '%n' placed in non-group symbol '%n'**

An attempt was made to place a section in a non-group type symbol

**Section '%n' placed in two different groups**

Section is placed in different groups

**Section '%n' placed in unknown group symbol #h**

There is an internal error in the object file

**Section '%n' must be in one of groups text, data or BSS**

When producing ST format code

**Symbol '%n' multiply defined**

New definition of previously defined symbol

**Symbol '%n' not defined**

Undefined symbol

**Symbol '%n' placed in non-section symbol #h**

There is an internal error in the object file

**Symbol '%n' placed in unknown section symbol #h**

There is an internal error in the object file

**Symbol in COFF format file has unrecognised class**

COFF format files are those produced by Sierra C cross compiler, etc.

**Timed out sending data to target**

Target not responding or offline

**Too many file names specified**

Too many parameters on command line

**Too many modules to link**

Maximum of 256 modules may be linked

**Too many symbols in COFF format file**

COFF format files are those produced by Sierra C cross compiler, etc.

**Unable to open output file**

Could not open specified output file

**Undefined symbol in COFF file patch record**

COFF format files are those produced by Sierra C cross compiler, etc.

**Unit number must be in range 0-127**

**Unknown option /%c**

On Command line

**Unknown processor type '%s'**

Could not recognise target processor type

**Unrecognised relocatable output format**

/r option on command line

**User pressed Break/Ctrl-C**

Linking aborted by user

**Value (%1) out of range in instruction patch**

Value to be patched in is out of range

**WORKSPACE command can only be used when downloading output**

## Psylib Error Messages

### Librarian Messages:

**Cannot add module : it already exists**

Module may only appear in a library once

**Could not create object file**

Error creating object file when extracting

**Could not create temporary file**

Error creating temporary file

**Could not open/create**

System error opening file

**Error reading library file**

System error occurred reading file

**Error writing library file**

System error writing file, probably disk full

**Incorrect format in object file**

Error in object file format - re-build it

**No files matching**

No object files matching the specifications were found

**No library file specified**

**No object files specified**

**No option specified**

An action option must be specified on the command line

**Unknown option /**

On Command line, option not recognised



#	5-6
\$	5-6
&	5-5
\	3-6,5-6, 5-7
{	5-7
<>	3-11,5-4
/	2-2,3-11
@	2-2
-	3-11
!	3-11
\$	3-4
%	3-11
&	3-2, 3-11
()	3-11
*	3-7, 3-11,5-7
.	3-3
:	3-3
;	3-2
?	3-3
@	3-3, 3-7, 3-8
^	3-11
_	3-3
_RS	3-7
_filename	3-7
~	3-11
<	3-11
<<	3-11
<=	3-11
>	3-11
>=	3-11
>>	3-11
?	4-23
_RS	4-11
{	4-7
=	3-11,4-5, 4-38
=?	4-38
8 bit	A-1
16 bit slot	1-4
32 bit evaluator	1-15
64 bit data	1-15
<b>A</b>	
About The System	I-ii
adapter cartridge	I-i
Adapter firmware	I-iii
Adding A Watch	10-57
Additional hardware	1-2
Additional Notes	1-15
ALIAS	3-15
Alignment	3-10
Alignment	4-20
Alignment	8-4
Alternate Numeric	9-4
Anchoring a Pane	10-78
array	10-52
Environment Variable	2-4
Assembler	
Running with Brief	A-4
AUTOEXEC.BAT	A-1, A-2, A-4
Assembler	
Options	9-3, 9-5
Warning Messages	9-6
Assembler	
Command File	2-2
Error Messages	2-5
User Termination	2-4
Assembler	
Comment Lines	3-2
Constants	3-4, 3-7
Continuation Lines	3-2
Functions	3-9, 3-10
Operators	3-11
Assembler, White Space	9-5
Assigning Variables	10-54
Assignment Directives	4-2
Automatic Even	4-13, 4-14, 4-15
<b>B</b>	
Beta Test Scheme	10-6
boot block	1-15
Boot header block	1-2
Break at Point	10-66
Break if expression is true	10-66
Breakpoints	10-64
Brief	I-v
Brief	A-5
Macros	A-5
BSS	8-2
<i>buffer size</i>	A-1
<b>C</b>	
C++ compiler	I-ii
<i>card address</i>	A-1, A-2
cartridge emulation RAM	I-i
Cartridge port	I-iii
CASE	4-31
Case Sensitivity	9-4
<i>channel</i>	A-1
Character Constants	3-4
Checking the Installation	1-12
checksum values	1-15
Command Lines	
RUN	A-3
CNOP	4-20,8-4
CPE Files	A-3
Command Files	
PSYLINK	11-4
Command Lines	
PSYLINK	11-2
Command Files	
Assembler	2-2
Command Lines	
Assembler	2-2
Command Lines	
PSYMAKE	14-2

- COMMAND.COM..... 14-5  
 Compact cartridge..... I-iii  
 completion..... 10-58  
 Configuring Your SCSI Card..... 10-9  
 Connecting Cable..... 1-1  
 Constants  
   Character..... 3-4  
   Integer..... 3-4  
   Location Counter..... 3-7  
   Special..... 3-6  
   Time and Date..... 3-6  
 Continuation Lines  
   Assembler..... 3-2  
 Continual Update Rate..... 10-63  
 CPE File Properties..... 10-22  
 cross development system..... I-i
- D**
- DATA..... 4-17  
 DATASIZE..... 4-17  
 Date Constants..... 3-6  
 Debug stub code..... 1-14  
 Debug vector..... 1-14  
 Decrease Index..... 10-49  
 DEF..... 4-27  
 Deleting A Watch..... 10-62  
 Development systems..... I-iv  
 DISABLE..... 3-13  
 dip switches..... 1-5, 1-6  
 DMA Channel..... 1-6, A-1, A-2  
 DMA number..... A-1  
 DOS..... A-2  
 DO..... 4-34  
 DOS..... I-v
- E**
- Edit breakpoint..... 10-65  
 Editing A Watch..... 10-61  
 ELF libraries..... 1-16  
 ELF object file format..... 1-16  
 ELFCONV - Library Converter Program..... 1-16  
 Elfconv errors..... 1-17  
 ELSE..... 4-29  
 ELSEIF..... 4-29  
 END..... 4-29  
 ENDC..... 4-29  
 ENDCASE..... 4-31  
 ENDIF..... 4-29  
 ENDM..... 5-2  
 ENDR..... 4-32  
 ENDW..... 4-33  
 Environment Variables..... 2-4  
 EQU..... 4-3  
 EQUR..... 4-8  
 EQUUS..... 4-6  
 Error Messages  
   Assembler..... 2-5  
 Expand/Collapse..... 10-49  
 Expanding Or Collapsing A Variable..... 10-55
- Expressions  
   Constants..... 3-4  
   Functions..... 3-9, 3-10  
   Operators..... 3-11
- F**
- Edit breakpoint..... 10-65  
 Editing A Watch..... 10-61  
 ELF libraries..... 1-16  
 ELF object file format..... 1-16  
 ELFCONV - Library Converter Program..... 1-16  
 Elfconv errors..... 1-17  
 ELSE..... 4-29  
 ELSEIF..... 4-29  
 END..... 4-29  
 ENDC..... 4-29  
 ENDCASE..... 4-31  
 ENDIF..... 4-29  
 ENDM..... 5-2  
 ENDR..... 4-32  
 ENDW..... 4-31  
 Environment Variables..... 2-4  
 EQU..... 4-3  
 EQUR..... 4-8  
 EQUUS..... 4-6  
 Error Messages  
   Assembler..... 2-5  
 Expand/Collapse..... 10-49  
 Expanding Or Collapsing A Variable..... 10-55
- Expressions  
   Constants..... 3-4  
   Functions..... 3-9, 3-10  
   Makefiles..... 8  
   Operators..... 3-11
- G**
- game cartridge..... 1-2  
 game image..... 1-15  
 GLOBAL..... 9-11, 11-6  
 GNU C Program  
   Example Linker Command File..... 11-8  
 Gnu-C..... I-i
- H**
- Hardware interrupt..... 1-15  
 hardware required..... I-I  
 HEX..... 4-16  
 host PC..... A-1
- I**
- IBM..... I-v  
 IEEE32..... 4-17  
 IEEE64..... 4-17  
 INCBIN..... 4-25  
 INCLUDE..... 4-23  
 Increase Index..... 10-51  
 Installing Development Software..... 1-19  
 Installing The Debugger..... 10-4  
 Installing the GNU 'C' Software..... 1-11

Installing The Hardware.....	1-2	ORG.....	4-19 ,8-2
INSTR.....	6-4	<i>P</i>	
Interface Board.....	A-1	Pane Type.....	10-41
<i>intnum</i> .....	A-1	PC Board.....	1-1
IO Address.....	1-7	PC Interface	
IRQ Number.....	A-1	Installation .....	1-5
<i>L</i>		PC Interface board.....	1-5
Labels .....	10-76	PC memory.....	A-1
Labels		PC Processor.....	1-7
Format.....	3-3	PC Software	
Symbols.....	3-3	pointer.....	10-52
Labels		POPO.....	9-7
Local.....	7-1	Power supply.....	1-3
Launching The Debugger.....	10-13	Project.....	10-19
LED .....	1-3	PSYLIB librarian program.....	1-16
Librarian .....	<i>See</i> PSYLIB	PSYLIB.....	12-1
Library conversion.....	1-16	Command Line.....	12-2
library tool.....	1-16	PSYLINK.....	11-1
Linker .....	<i>See</i> PSYLINK	Command Line.....	11-2
LIST .....	9-7	PSYMAKE.....	14-1
LOCAL.....	7-4	Command Line.....	14-2
Local Labels		Makefile.....	14-3
Within Modules.....	7-3	PUBLIC.....	11-7
Local Labels		PUBLIC.....	9-10
Descopse.....	9-4	PURGE.....	5-10
Signifier.....	9-5	PUSHO.....	9-7
Location Counter.....	3-7	PUSHP.....	5-9
<i>M</i>		PUSHS.....	8-6
MACRO.....	5-2	<i>R</i>	
Macros.....	5-10	RADIX.....	3-12
Continuation Lines.....	5-4	REF.....	4-26
Control Characters.....	5-6	REGS.....	4-36
Entire Parameter .....	5-6	Register Names.....	10-75
Extended Parameters.....	5-6	REPT .....	4-32
Integers to Text.....	5-5	RSRESET.....	4-11
Parameter Type.....	5-11	RSSET .....	4-10
Parameters.....	5-3	RUN.....	A-3
Unique Labels.....	5-5	Command Line.....	A-3
MEXIT .....	5-2	Run to cursor.....	10-45
MODEND.....	7-3	R4300 assembler.....	I-i
MODULE.....	7-3	<i>S</i>	
<i>N</i>		Saturates access to cartridge ram.....	A-2
NARG.....	5-7	Scope of Local Labels.....	7-2
Nintendo 64 libraries.....	1-15	SCSI bus.....	1-8
NOLIST.....	9-7	SCSI Card.....	1-2
<i>O</i>		SCSI ID.....	1-8
OBJ.....	4-21	SCSI ID duplication.....	1-8
OBJ.....	8-3	SCSI parallel link.....	I-ii
OVER.....	8-3	SCSI Termination Power.....	1-7
OBJEND.....	4-21	SECT .....	8-7
Obtaining Releases And Patches.....	10-5	Simple Name Completion.....	10-58
OFFSET.....	8-7	Structuring the Program.....	8-1
On-line Help Available For The Debugger.....	10-3	security chip.....	1-2
Operator Precedence.....	3-11	SET.....	4-4
Operators .....	3-11	setcsum.exe.....	1-15
		SGI workstation.....	1-16

SHIFT.....	5-7	Toolbar Icons.....	10-29
SIZE .....	8-2	Traversing An Index.....	10-56
Specifying Binary File Properties.....	10-24	TYPE .....	5-11
Specifying Symbol File Properties.....	10-23	Typecasts and Typedefs.....	10-75
Step Into command.....	10-67	<i>U</i>	
Step Over command.....	10-68	Unit toolbar.....	10-17
STRCMP.....	6-3	UNTIL .....	4-34
String Manipulation Functions.....	6-1	<i>W</i>	
STRLEN.....	6-2	Warnings, Assembler Messages.....	9-5
SUBSTR.....	6-5	White Space, Assembler.....	9-5
Syntax of Local Labels.....	7-2	WHILE.....	4-33
<i>T</i>		WORD.....	8-2
Target Interface.....	1-1	<i>X</i>	
Target interface.....	I-iii	XDEF.....	9-10, 11-7
TBIOS2.....	A-2	XREF.....	9-10, 11-7
TBIOS2 Command Line.....	A-1	<i>Z</i>	
TBIOS2.COM.....	A-1	Zilog Numbers.....	9-4
TSR program.....	A-1		
test program.....	1-15		
Time Constants.....	3-6		
TLB .....	1-15		