July 1995
Order Number: 312853-005

Paragon™ System
Source Code Product
Release 1.3
User's Guide

Intel® Corporation

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

LIMITED RIGHTS

Preface

This manual describes how to use the Paragon™ System Source Code Product (PSCP), Release 1.3.

This manual assumes you have root privileges on the Sun4/SunOS 4.x system. The manual provides you with enough information to install the PSCP, build the operating system software, and install the software you've built onto a Paragon system.

Organization

The manual is organized as follows:

Chapter 1	Describes how to install the PSCP.
Chapter 2	Describes how to build the system software.
Chapter 3	Describes how to test the build.
Chapter 4	Describes how to modify sources.
Chapter 5	Describes how to check for errors.
Appendix A	Lists source file changes from the R1.3 release build.
Appendix B	Lists the build tools provided with the PSCP.
Appendix C	Provides a map of the directory tree structure.

Notational Conventions

This manual uses the following notational conventions:

Bold Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.

Italic Identifies variables, filenames, directories, processes, user names, and writer annotations in examples. Italic type style is also occasionally used to emphasize a word or phrase.

`Plain-Monospace`
Identifies computer output (prompts and messages), examples, and values of variables. Some examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style and flush with the right margin.

Bold-Italic-Monospace
Identifies user input (what you enter in response to some prompt).

`%` At the start of a line, indicates input typed by a non-privileged user.

`#` At the start of a line, indicates input typed by user *root* (a privileged user) on a Paragon.

DS# At the start of a line, indicates input typed by user *root* (a privileged user) on a diagnostics station.

Applicable Documents

For more information, refer to the *Paragon™ System Technical Documentation Guide*.

Comments and Assistance

Intel Scalable Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

Intel Corporation Italia s.p.a.
Milanofiori Palazzo
20090 Assago
Milano
Italy
1678 77203 (toll free)

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Intel Japan K.K.
Scalable Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com
(Internet)

Table of Contents

Appendix A
Source Changes from R1.3

Appendix B
Build Tools

Appendix C
PSCP Directory Structure

Installing the PSCP 1

This chapter describes how to install the Paragon™ System Source Code Product (PSCP) onto a system and build a new operating system.

Legal Requirements

Using the PSCP is subject to the terms of the Intel Source License Agreement between you and Intel Corporation. After installation of the PSCP, you should also read the file located in */etc/copyright*. This file outlines the legal limitations you have agreed to when purchasing the PSCP and contains copyright information about the software you have purchased.

System Requirements

The PSCP will build Release 1.3 of the operating system on a Sun/SunOS 4.x system.

Installing the PSCP requires about 90M bytes of available writable disk space. Each successive phase of the build and installation process requires additional disk space.

Table 1-1 gives dependencies, approximate disk space requirements for source code and build products, and build times for the individual build components listed in the next section. The times listed in the table may vary.

Table 1-1. Disk and Time Requirements

Component	Dependencies	Initial Disk Space Requirements (Megabytes)	Total Disk Space Required For Building (Megabytes)	Hours to Build (Sparc2)
Build Tools and Files	-	14	20	-
Microkernel	-	15	58	2
Server/Emulator	Microkernel	16	47	3
Commands/Libraries Build	Microkernel, Server	41	162	3
Export Tree	Microkernel, Server, Commands/Libraries	-	84	.5
RAID Build	Installation of Commands/Libraries	3	12	.5
RAID Install	RAID Build	-	2	.05
Installable Archives	(All of the above)	-	75	.05
Total	-	89	460	9.1

Components

The PSCP consists of the following components:

Build Tools and Files	Compilers, **make**, and other tools and files used in building the PSCP.
Microkernel	Sources and makefiles for building the kernels, which will be installed in the */mach_servers* directory as *mach_kernela* and *mach_kernelb*. Also builds debuggable kernels, installed as *mach_kernela.db* and *mach_kernelb.db*.
Server/Emulator	Sources and makefiles for building the server and emulator, which are installed in the */mach_servers* directory as *startup* and *emulator*, respectively. Also builds a "lite server", installed as *startup.compute*.
commands/libraries	Sources and makefiles for building the OSF* set of commands and libraries, as modified by Intel. This and the previous two build environments were developed at OSF.

	Although these three environments are not really a unified whole, there are dependencies among them, as shown in Table 1-1.
RAID utilities	Builds the set of RAID utilities, which will be installed mostly into the */sbin* and */usr/sbin* directories.
Exports Tree	Installable image of the entire file system with the correct permissions, owners, and groups.
Release Tree Installable Archives	Creates archives which may be installed onto a Paragon system.

Installing from Tape

The PSCP is delivered on a QIC-150 tape. You can install it on any file system on which there are 90M bytes of free space. An additional 371M bytes are needed for a complete build. If you use an NFS file system for the installation, it must be writable by *root*.

Throughout this document the directory into which the PSCP is installed is referred to as *$PBE*.

The following example describes how to install the PSCP, assuming the PSCP tape is in tape drive /dev/rst0 on a Sun workstation, in */home/PBE*:

```
% su
# mkdir /home/PBE
# PBE=/home/PBE; export PBE
# cd $PBE
# umask 002
# tar xvfp /dev/rst0
```

The commands shown in this example would install everything needed for the build process in */home/PBE*. Some other directory may be substituted for */home/PBE* in the above example if you wish to use another location for *$PBE*.

Setting up the Environment

As shipped, the contents of the PSCP are owned by *root*. After reading in the tape, set the owner and group to the appropriate names for your site. For example, if the owner and group should be *swdev* and *sweng*, you would enter the following as *root*:

```
# cd $PBE/..
# chown -R swdev.sweng PBE
# chown root $PBE/Bld/Tools/bin/release
```

The file *$PBE/Bld/Tools/bin/release* must be owned by root.

You may also want to alter the default value of *BUILD_VERSION_STRING* as set in *$PBE/Config.mk*. As installed, this value will match the string used in the release whose source code you are building (in this release, *R1_3*). If you wish, you can change it to whatever you like. This string is displayed by running the **what** command on any binary built with this version of the PSCP.

For example, the following command:

```
% what /usr/bin/csh
```

when run on a R1.3 system normally displays:

```
csh R1_3-01 Fri April 21 15:25:31 1995
```

If you were to change the value of *BUILD_VERSION_STRING* in *$PBE/Config.mk* file to *REBUILD_R1_3*, then build the commands and install them on a Paragon system, the same **what** command would display something like the following:

```
csh REBUILD_R1_3 Wed Jun 14 12:32:51 1995
```

You can use the *BUILD_VERSION_STRING* variable, along with the **what** command, to uniquely identify versions of binaries.

Add *$PBE/Bld/Tools/bin* to your execution path. If you are using *csh*, enter the following:

```
set path = ($PBE/Bld/Tools/bin $path)
```

If you are using *sh*, enter the following:

```
PATH=$PBE/Bld/Tools/bin:$PATH; export PATH
```

The settings for *PBE* and *PATH* must be in place for the entire build.

Software Bill of Materials

A software bill of materials of the source files included in the PSCP is installed in *$PBE/BomFile* during PSCP installation. For each file in the release, the bill of materials gives the name of the file followed by its CVS revision number. There are over 9,700 source files in the PSCP.

Removing the PSCP

To remove the PSCP from your system, type the following:

```
# rm -rf $PBE
```

Building the System Software 2

This chapter describes how to build Release 1.3 of the operating system using the PSCP. It assumes that the PSCP has been installed on a Sun4 system.

Disk and Time Requirements

Compiling and linking all the components of the PSCP takes about 9 hours and requires about 210M bytes of disk space beyond that consumed when the PSCP was installed. Installing the software and creating tar files requires an additional 161M bytes of disk space. A breakdown of requirements for building the individual components is given in Table 1-1 on page 1-2.

Building Everything

Building all parts of the PSCP (microkernels, servers, emulator, commands/libraries, RAID utilities, and installable tar files) can be done in either of two ways, depending on whether or not you wish to build as an ordinary user. The install phase, where the Exports directory is populated, must be done as *root*.

As *root*, you would enter the following commands:

```
# cd $PBE
# pmake -N all
```

As an ordinary user, you would enter the following commands:

```
% cd $PBE
% pmake -N build_all
```

Builds the kernels, servers, emulator, commands, and libraries.

```
% su
```

```
# pmake -N install_all
```
Installs the kernels, servers, emulator,commands, and libraries in *$PBE/Exports* and creates *$PBE/Exports/sysfiles.tar*, which is needed for the RAID utility build

```
# exit
% pmake -N -C raid build_all
```
Installs sysfiles.tar in *$PBE/bld/Compilers* and builds the RAID utilities.

```
% su
# pmake -N -C raid install_all
```
Installs RAID utilities in *$PBE/Exports*.

```
# pmake -N tarfiles
```
Creates *usr.tar*, *root.tar*, and *mach_svr.tar* in *$PBE/Release*.

Log files are written into *$PBE/Bld/logs*. The log files written by the **pmake -N build_all** command include the following:

mach_kernela.build	Log from building *mach_kernela* (ASMP-bigpkts)
mach_kernelb.build	Log from building *mach_kernelb* (ASMP+nicb)
mach_kernela.db.build	Log from building *mach_kernela.db* (ASMP-bigpkts+assert)
mach_kernelb.db.build	Log from building *mach_kernelb.db* (ASMP+nicb+assert)
startup.build	Log from building server (STD+WS)
startup.compute.build	Log from building server (LITE+WS)
emulator.build	Log from building emulator
commands.build	Log from building commands and libraries

The log files written by the **pmake -N install_all** command include the following:

commands.install	Log from installing commands and libraries
postinstall	Log from executing postinstall. Postinstall creates required links and sets permissions for some files.

The log files written by the **pmake -N -C raid build_all** command include the following

raid.build	Log from build the RAID utilities.
Raid.Build.Finished.Successfully	Status file indicating that the RAID build succeeded. This file must be removed before rebuilding the RAID utilities.

Log files are also created when any part of the PSCP is built, except for a component of the commands/libraries. When a new log file is created, the old log file is not overwritten. Instead, a numeric suffix is appended to the original file name.

Building Parts of the PSCP

This section describes how to build each part of the PSCP.

Some parts are dependent on others having already been built. For example, if you want to build a server, you must first build a kernel. See Table 1-1 on page 1-2 for dependency information.

Building the Microkernel

You can specify which kernel to build or build all four kernels (*mach_kernela*, *mach_kernelb*, *mach_kernela.db*, and *mach_kernelb.db*)

To build a single kernel, enter the following command:

```
% cd $PBE
% pmake -N -C kernels kernel_name
```

To build all four kernels, you would enter the following command:

```
% cd $PBE
% pmake -N -C kernels build_all
```

The path name of the kernel(s) built is:

$PBE/kernels/obj/i860_mach/mk/kernel/<Config>/mach_kernel

<Config> is one of the following:

ASMP+nicb
ASMP-bigpkts
ASMP+nicb+assert
ASMP-bigpkts+assert

Building the Servers and Emulator

You can build the standard server, the lite server, the emulator, or all three targets. To build a single component, enter the following command:

```
% cd $PBE
% pmake -N -C servers component_name
```

component_name can be startup, startup.compute, or emulator

To build all three components, you would enter the following command:

```
% cd $PBE
% pmake -N -C servers build_all
```

The standard server executable is:

$PBE/servers/svr/obj/i860_mach/latest/server/STD+WS/vmunix

The lite server executable is:

$PBE/servers/svr/obj/i860_mach/latest/server/LITE+WS/vmunix

The emulator executable is:

$PBE/servers/svr/obj/i860_mach/latest/emulator/STD+WS/emulator

Building the Commands and Libraries

To build the commands and libraries enter the following:

```
% cd $PBE
% pmake -N -C cmds build_all
```

Commands and libraries built will be found in *$PBE/cmds/obj/ipsc860/<PATH_TO_COMMAND>*, where *<PATH_TO_COMMAND>* is the directory under *$PBE/cmds/src* in which sources for the command are found. For example, sources for the **df** command are found in *$PBE/cmds/src/usr/bin/df*, and the **df** executable will be found after the build in the *$PBE/cmds/obj/ipsc860/usr/bin/df* directory.

Installing the Commands and Libraries

Before the RAID utilities can be built, you must install the commands and libraries. Become *root*, and enter the following commands:

```
% su
# cd $PBE
# pmake -N -C cmds install_all
```

This command produces the log files *$PBE/Bld/logs/commands.install* and *$PBE/Bld/logs/postinstall*. It installs the commands and libraries in *$PBE/Exports*, sets the permissions correctly, and creates *$PBE/Exports/sysfiles.tar*. (*sysfiles.tar* is needed for the RAID build.)

Building the RAID Utilities

You must install the commands before building the RAID utilities. To build the RAID utilities, enter the following:

```
% cd $PBE
% pmake -N -C raid build_all
```

After this runs to completion, the RAID utilities will be found in directories under *$PBE/raid/src/raid/oemsrc/src*. For example, the **ace** binary will be found in the directory *$PBE/raid/src/raid/oemsrc/src/Ace*.

Installing the RAID Utilities

Before making installable tar files, you must install the RAID utilities in *$PBE/Exports*. Become *root*, and enter the following commands:

```
% su
# cd $PBE
# pmake -N -C raid install_all
```

This command installs the RAID utilities in *$PBE/Exports* and sets the permissions correctly.

Creating the Archives

After a build has successfully completed and the *$PBE/Exports* area has been populated, the next step in the process is to create **tar** archives that can be installed from the diagnostic station or a remote workstation. You can create release archives as user *root* by entering the following:

```
# cd $PBE
# pmake -N tarfiles
```

This command creates three **tar** files in the *$PBE/Release* directory: *mach_svr.tar, root.tar,* and *usr.tar*. They contain, respectively, the contents of the */mach_servers*, /, and */usr* directories on the target system.

After this step you will have created archives containing the complete operating system, which can be installed to test your newly built software. If you wish, the contents of the following directories can be deleted at this time to conserve space:

$PBE/Exports
$PBE/mk/obj/i860_mach/mk
$PBE/servers/svr/obj/i860_mach
$PBE/cmds/obj/ipsc860

Removing the *servers/svr/obj* directory will require you to rebuild at least one server before the commands and libraries can be rebuilt.

Testing the Build 3

Test the software you've built with the PSCP either by installing a completely new operating system, or by choosing the component(s) you want to test and installing them one at a time into system locations on the Paragon system. This chapter discusses both approaches.

Disk Requirements

Installing built software overwrites existing system software. If you are installing portions of the operating system you can rename existing files with the **mv** command before installing your test versions.

If you install archives of the entire operating system , the entire existing operating system will be overwritten. In this case, you can return to the official Release 1.3 software by re-installing from the cartridge tapes that were shipped with your system. See the *Paragon™ System Software Installation Guide* for installation information.

As an alternative, you can install the software you have built onto an alternate boot disk, then boot from that disk, rather than the normal boot disk. See the *Paragon™ System Administrator's Guide* for information about installing from a boot disk.

Installing Piecemeal

Individually built pieces can be installed by copying them out of the build tree into their destinations in the Paragon system's file system.

For example, to test a newly built version of the **df** command, do the following. Note that you must be *root* to write into */usr/bin*. This example assumes that the *$PBE* directory has been mounted on your Paragon system. If this is not the case, use **ftp** to copy the file to your Paragon system.

```
% su
# mv /usr/bin/df /usr/bin/df.dist
# cp $PBE/cmds/obj/ipsc860/usr/bin/df/df /usr/bin
```

If necessary, you should reset ownerships and permissions of the new version of **df** to match those of the original version. The new **df** will then be used by all users by default.

If the program to be tested is a process that is normally running as part of the operating system (**startup, emulator**, etc.), you have to reboot the system to test the new software. For example, to test a new version of the emulator enter the following. This example assumes that the *$PBE* directory has been mounted on your Paragon system. If this is not the case, use **ftp** to copy the file to your Paragon system.

```
% su
# mv /mach_servers/emulator /mach_servers/emulator.R1.3
# cd $PBE/servers/svr/obj/i860_mach/latest/emulator/STD+WS
# cp emulator /mach_servers
```

You must now reboot the system to try out the new emulator. Refer to the *Paragon™ System Administrator's Guide* for information about rebooting the system. When you come back up, you'll be running with the new version of the emulator.

If you want to test a new microkernel you have built, you'll also have to install it on your diagnostic station. The new kernel should be copied to the */usr/paragon/boot* directory on your diagnostic station.

If you have built a non-debuggable kernel such as *mach_kernela* (ASMP-bigpkts) or *mach_kernelb* (ASMP+nicb), then the kernel should be installed as */usr/paragon/boot/mach_kernela* or *mach_kernelb* and also as *mach_kernel.md* on the diagnostic station. If you have built a debuggable kernel *mach_kernela.db* (ASMP-bigpkts+assert) or *mach_kernelb.db* (ASMP+nicb+assert), then it should be installed as */usr/paragon/boot/mach_kernela.db* or *usr/paragon/boot/mach_kernelb.db* on the diagnostic station.

The *mach_kernel.md* that shipped with R1.3, is a stripped version of *mach_kernela.*

Installing the Entire Operating System

To completely rebuild the operating system and install everything from the archives you built, copy the archives you placed in *$PBE/Release* onto your diagnostic station and install them from ramdisk. To do this, log onto your diagnostics station as user *root*, then enter the following:

```
DS# cd /u/tmp
DS# cp /etc/hosts .
```

Establish an **ftp** connection to the Sun system on which your software was built.

```
DS# ftp Sun_hostname
```

Log in to **ftp** as user *root*. Throughout this **ftp** session, *PBE* is the value of *$PBE*.

```
ftp> cd PBE/Release
ftp> type binary
ftp> get mach_svr.tar
ftp> get root.tar
ftp> get usr.tar
ftp> lcd /usr/paragon/boot
ftp> cd PBE/Exports/paragon/mach_servers
```

Perform the next step only if you have built a non-devuggable kernel (ASMP-bigpkts or ASM+nicb).

```
ftp> get mach_kernelx
```

x is a or b.

Perform the next step only if you built a debuggable kernel (ASMP-bigpkts+assert or ASMP+nicb+assert).

```
ftp> get mach_kernelx.db
ftp> quit
```

x is a or b.

Now you can follow the procedure outlined in the *Paragon™ System Software Installation Guide.* However, skip the instructions related to obtaining the installation tar files, because you have already copied them over from your build system and do not need to install them from tape.

Modifying Sources 4

This chapter describes how to change source files included in the PSCP and how to build a reconfigured kernel or server.

Changing Source Files

Source files can be modified using **vi** or any text editor. In most cases, changing a source file automatically causes those object files that depend on it to be recompiled. This is generally true in the command/libraries and RAID utilities build environments, and is usually true for the microkernel and server/emulator environments.

If you suspect that object files are not being regenerated when they should be, you can delete the object trees and start a build from scratch. For example, in a server build you would enter the following:

```
% rm -rf $PBE/servers/obj/i860_mach/latest/*
```

This completely removes the object tree, so the next time you run **make** in the source tree, all object files are regenerated.

Error Checking 5

While most of the software in the PSCP will build without any problems, it is possible that you may encounter errors during the software build and installation process. This chapter discusses what happens when a build error occurs, and what can be done to discover the problem.

Action on Errors

Builds of all components of the PSCP halt immediately if a fatal build error occurs, except for the Commands/Libraries build. After running the **pmake -N build_all** command, you can run the **tail** command on the log files to see if they all built correctly. If the end of the log file for a specific part of the PSCP build ends with an error message, the build failed. Otherwise it succeeded.

The exception to this is the Commands/Libraries build. This build does not stop on fatal errors. When invoked with the **-k** flag and the **build_all** option, **cmake** uses a multipass algorithm that can try four or five times to build the same command. As output from **cmake** contains very long lines that break some common text editors, such as **vi**, a tool to examine output from a Commands/Libraries build has been provided, called **geterrs**.

Using *geterrs*

The **geterrs** tool is located in *$PBE/Bld/Tools/bin*. It can be used to get a summary of those commands for which builds failed, or to see the entire build output for each command whose build failed. To get a summary of those commands for which builds failed, enter the following:

```
% geterrs -s $PBE/Bld/logs/commands.build
```

This example assumes the log file for a run of **cmake** was the *CmdsOutput* file. Output of **geterrs** would look similar to the following:

```
>>>> [ /usr/ccs/lib/migcom ]: Error on line 7371 <<<<
>>>> [ /usr/lib/diction ]: Error on line 8150 <<<<
>>>> [ /usr/lib/diction ]: Error on line 8159 <<<<
>>>> [ /usr/lib/diction ]: Error on line 8162 <<<<
>>>> [ /usr/bin/qmgr ]: Error on line 11262 <<<<
>>>> [ /usr/bin/awk ]: Error on line 11398 <<<<
>>>> [ /usr/bin/nslookup ]: Error on line 130
```

Each command whose build fails is delineated by brackets and followed by the line of the log file on which the error message occurred.

To see the entire build output for each command whose build failed, enter the following:

```
% geterrs $PBE/Bld/logs/commands.build
```

This command would produce something similar to the following:

```
>>>> [ /usr/ccs/lib/migcom ]: Error on line 7371 <<<<
[ /usr/ccs/lib/migcom ]
makepath migcom/. && cd migcom && exec make MAKEFILE_PASS=BASIC
dopass_all
migcom: created directory
yacc -d ../../../../../../src/usr/ccs/lib/migcom/parser.y
mv -f y.tab.c parser.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
parser.c
if [ -f y.tab.h ]; then mv -f y.tab.h parser.h; fi
lex ../../../../../../src/usr/ccs/lib/migcom/lexxer.l
"../../../../../../src/usr/ccs/lib/migcom/lexxer.l", line 246:
(Error) Lex driver missing, file /usr/lib/lex/ncform
1304/10000 nodes(%e), 3452/25000 positions(%p), 429/5000 (%n),
26137 transitions
, 897/1000 packed char classes(%k), 2181/20000 packed
transitions(%a), 1963/3000 output slots(%o)
*** Exit 1
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/migcom.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/error.c
```

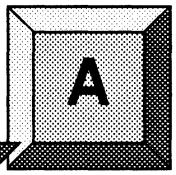
```
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/string.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/type.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/routine.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/statement.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/global.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/header.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/user.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/server.c
icc -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF -Di860 -D__i860__
-nostdinc `genpath -I.` -I- `genpath `
-I/vol/PBE/ParagonSrcDirs/cmds/export/ipsc860/usr/include
../../../../../../src/usr/ccs/lib/migcom/utils.c
`dopass_all' not remade because of errors

etc. [ similar output for the other commands ]. . .
```

The 'Exit' above is on line 7371, as indicated by the output of **geterrs -s**. In this case the problem was caused by a missing file in */usr/lib/lex*.

See the *Paragon™ System Source Code Product Release 1.3 Release Notes* that came with the PSCP for a list of known errors in this release.

Source Changes from R1.3



The only source changes between the R1.3 PSCP and R1.3 were updates to the copyright information. The original sources are not included in this product.

Build Tools B

This appendix lists the build tools delivered with the PSCP and provides a brief description of their use.

These tools are installed in *$PBE/Bld/compilers*, *$PBE/Bld/Tools/bin*, and *$PBE/Bld/Tools/lib*.

$PBE/Bld/Compilers

This directory contains the R4.5 Sun4/SunOS 4.x C cross-development tools, libraries, and objects.

$PBE/Bld/Tools/bin

ar

Executes **ar860**, the system archiver.

cc

Executes the R4.5 C compiler, after adding these compiler switches:

-Mnoperfmon -Mnodebug -Mnostdinc -Di860

This version of **cc** is used to compile programs run during the build process.

cc860

A shell script used in the kernel build. Depending on its parameters, it calls either **icc** or **cpp860** from the *$PBE/Bld/Tools/bin* directory.

cmake

Version of **make** used to build the commands and libraries.

cpath

A binary program used to translate the environment variable *CPATH* (set in *$PBE/Bld/include/Raid.mk* and modified by *make* and *cmake)* into a line of the form:

```
-I/usr/include -I/usr/local/include
```

which can then be passed to the compiler.

cpp860

Executes *$PBE/Bld/lib/cpp*, the PSCP C preprocessor.

ctab

File processing tool.

do.depends

Determines if **md** (make depends) needs to be run.

gcc

The **gcc** compiler.

gencat

Processes catalog files.

genpath

Generates command flags for build areas.

geterrs

A binary tool that may be used to extract error information from the output of a commands/libraries build. For example, to get a listing of those commands whose build failed, along with the line number in the log file on which the failure is printed, from logfile *$PBE/Bld/logs/commands.build*, enter:

```
% geterrs -s $PBE/Bld/logs/commands.build
```

To see all the output for commands that failed to build, enter the following:

```
% geterrs $PBE/Bld/logs/commands.build
```

icc

The compiler driver used by the PSCP in the server and commands/libraries builds. This version has been slightly modified from */usr/ccs/bin/icc* to support some non-standard compiler flags used in the operating system build process.

ld860

ld860 strips out some arguments to **ld** used in the PSCP build before calling **ld**, the system linker.

libloc

Initializes the list of known environments used by setlocale.

lpath

A binary program used to translate the environment variable *LPATH* (set in *$PBE/Bld/include/OS.mk* and modified by *make* and *cmake)* into a line of the following form:

```
-L/lib -L/usr/lib
```

This can then be passed to the linker.

make

Version of */usr/ccs/bin/make* used to build the kernel and server.

makepath

Creates intermediate directories along a path.

md

A binary program to generate *Makefile* dependency lists, used in the PSCP build process.

mig

Version of */usr/bin/mig* (Mach Interface Generator) used in the PSCP build process.

mkidinfo

A binary program that creates version string information that the PSCP *Makefiles* then embeds in built binaries. This can then be extracted with the **what** program.

perl

Practical Extraction and Report Language. Perl source code can be found in several locations on the Internet, including the following:

ftp://ftp.khoros.unm.edu/pub/perl/perl-4.036.tar.gz
ftp://prep.ai.mit.edu/pub/gnu/perl-4.036.tar.gz
ftp://metronet.com/pub/perl/source/perl4.036.tar.Z
ftp://ftp.celestial.com/pub/sco-ports/unix/perl-4.0.36.tar.gz

pmake

Parallel make utility.

ranlib

Dummy executable.

ranlib860

A shell script that does nothing, but must exist for the PSCP build process.

release

Program used during the installation of the commands and libraries.

wh

A binary program used by the PSCP to determine complete paths to executables and libraries.

xmkcatdefs

Makes message catalog definitions.

xstrip

A shell script that immediately exits. This script is necessary for the PSCP kernel build.

yacc

Parsing program generator.

$PBE/Bld/Tools/lib

cpp

The PSCP version of the C preprocessor, **cpp860**.

libcs.a

Library linked in with some of the PSCP build tools.

migcom

Compiler used by the **mig** program.

ncform

File needed by the **lex** program.

yaccpar

File needed by the **yacc** program.

PSCP Directory Structure

C

This appendix provides a map of the directory structure for the important directories used in the PSCP. A brief description of directory contents follows.

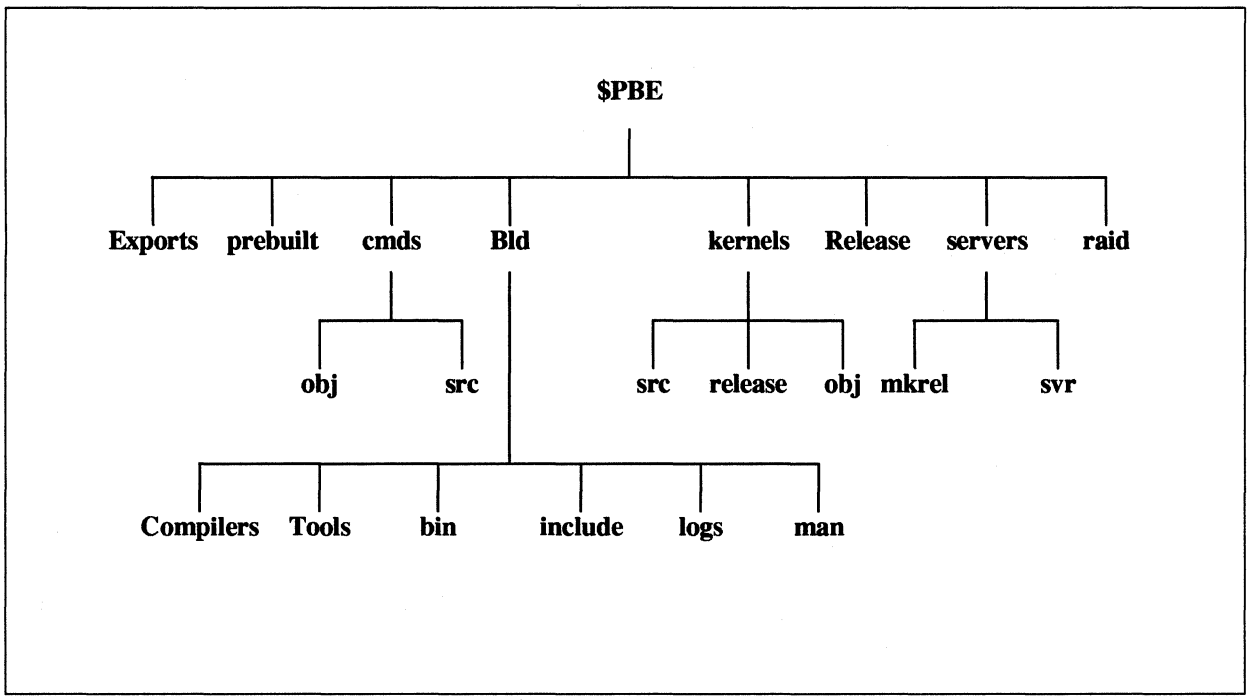


Figure C-1. PSCP Directory Structure

The following is a description of the directories shown:

$PBE	Root of the PSCP build tree
$PBE/Bld/Tools/bin	Tools used in PSCP build
$PBE/Exports	Root of PSCP exports tree
$PBE/Bld/Tools/lib	Files used in PSCP build
$PBE/Release	Repository for installable archives
$PBE/kernels	Root of kernel build tree
$PBE/cmds	Root of commands/libraries build tree
$PBE/servers	Root of server build tree
$PBE/raid	Root of RAID utilities source tree
$PBE/kernels/release	Kernel build products used in server build
$PBE/kernels/obj	Kernel build object files
$PBE/kernels/src	Kernel source tree
$PBE/cmds/obj	Commands build object files
$PBE/cmds/src	Commands/Libraries source tree
$PBE/prebuilt	Contains pre-built objects
$PBE/servers/mkrel	Link to kernel release tree for server build
$PBE/servers/svr	Base of server source and object trees