UNICOS/mk® Configuration Reference Manual

004–2603–002

New Features

UNICOS/mk® Configuration Reference Manual 004–2603–002

For the 2.0.5 release of UNICOS/mk, the following information was added to this book:

- In Chapter 7, Downing a PE, an informational note was added at the beginning of the chapter about the `renumber`(8) command.
- In Chapter 9, Disk Device and File System Configuration, section 9.5.4, The `/etc/config/rcoptions` File, descriptions were added about the two new `rcoptions` parameters, `DUMPCMPAPP` and `DUMPNPROC`, which support the new dump compression capability.

Record of Revision

Version	*Description*
1.2	August 1996 Draft printing.
1.2	October 1996 Updated draft printing.
1.3	November 1996 Updated draft printing to support UNICOS/mk 1.3.
1.3.1	December 1996 Updated draft printing to support UNICOS/mk 1.3.1. Procedures have been updated and general information has been clarified and added throughout.
2.0	January 1997 Updated draft printing to support SWS–ION release 2.0. Procedures have been updated and general information has been clarified and added throughout.
1.4.1	February 1997 Updated draft printing to support UNICOS/mk 1.4.1. Includes updates and information added for the SWS-ION release 2.0.
1.4.2	March 1997 Updated printing to support UNICOS/mk 1.4.2. Includes updated section on downing a PE.
1.5	April 1997 Updated printing to support UNICOS/mk 1.5. Includes new sections on configuration parameters and the remote mount facility.
1.5.1	May 1997 Updated printing to support UNICOS/mk 1.5.1.
1.5.2	June 1997 Updated printing to support UNICOS/mk 1.5.2.
1.6	July 1997 Updated printing to support UNICOS/mk 1.6.
1.6.2	August 1997 Updated printing to support UNICOS/mk 1.6.2.

2.0	October 1997 Updated printing to support UNICOS/mk 2.0.
2.0.2	January 1998 Updated printing to support UNICOS/mk 2.0.2.
2.0.3	April 1998 Updated printing to support UNICOS/mk 2.0.3.
2.0.4	November 1998 Updated printing to support UNICOS/mk 2.0.4.
2.0.5/002	October 1999 Updated printing to support UNICOS/mk 2.0.5.

Contents

Page

Page

Page

Page

Page

Figures

Tables

Preface

This publication documents the UNICOS/mk release running on Cray T3E systems. This manual provides information for system administrators who manage the operation of any Cray T3E system running the UNICOS/mk operating system.

Related Publications

The following documents contain additional information that may be helpful:

- *UNICOS/mk General Administration*
- *Tape Subsystem Administration*
- *UNICOS/mk Resource Administration*
- *UNICOS/mk Networking Facilities Administration*
- *UNICOS/mk Installation Guide for Cray T3E Series Systems*
- *SWS-ION Release Overview*
- *SWS Solaris Operating System and Devices Installation Guide*
- *SWS-ION Administration and Operations Guide*
- *Cray Scalable I/O Functional Overview*
- *Cray Scalable I/O Messages*

Obtaining Publications

The *User Publications Catalog* describes the availability and content of all Cray hardware and software documents that are available to customers. Customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a document, call +1 651 683 5907. SGI employees may send e-mail to `orderdsk@sgi.com`

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items (such as commands, files, routines, pathnames, signals, messages, programming language structures, and e-mail addresses) and items that appear on the screen.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers: 1 User commands 1B User commands ported from BSD 2 System calls 3 Library routines, macros, and opdefs 4 Devices (special files) 4P Protocols 5 File formats 7 Miscellaneous topics 7D DWB-related information 8 Administrator commands Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.

[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

The default shell in the UNICOS and UNICOS/mk operating systems, referred to as the *standard shell*, is a version of the Korn shell that conforms to the following standards:

- Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) Standard 1003.2–1992
- X/Open Portability Guide, Issue 4 (XPG4)

The UNICOS and UNICOS/mk operating systems also support the optional use of the C shell.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and part number of the document with your comments.

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `techpubs@sgi.com`

- Send a fax to the attention of “Technical Publications” at: +1 650 932 0801.
- Use the Feedback option on the Technical Publications Library World Wide Web page:

  `http://techpubs.sgi.com`

- Call the Technical Publications Group, through the Technical Assistance Center, using one of the following numbers:

  For SGI IRIX based operating systems: 1 800 800 4SGI

  For UNICOS or UNICOS/mk based operating systems or Cray Origin 2000 systems: 1 800 950 2729 (toll free from the United States and Canada) or +1 651 683 5600

- Send mail to the following address:

Technical Publications
SGI
1600 Amphitheatre Pkwy.
Mountain View, California 94043–1351

We value your comments and will respond to them promptly.

Introduction [1]

This manual provides background reference information to use when planning the configuration of a Cray T3E system running the UNICOS/mk operating system.

New Configuration File Features [2]

This chapter describes the new UNICOS/mk features that affect the UNICOS/mk configuration file. This chapter includes sections on the following features:

- The `#include` directive (UNICOS/mk 2.0.3 release)
- Single instance of the `mainframe_t` structure (UNICOS/mk 2.0 release)
- Miscellaneous configuration file changes

2.1 The `#include` Directive in UNICOS/mk 2.0.3

For the UNICOS/mk 2.0.3 release, a new feature has been added to the configuration file parser, `csp`(8), which drastically decreases the size of each configuration file. It is no longer required that header files be appended at the beginning of a configuration file. Instead, each header file can now be specified through an `#include` directive, which instructs the configuration file parser to search and read the specified file.

> **Note:** The new configuration file parser will still parse configuration files that have the include files appended to the beginning of the file. The ability to parse these types of configuration files will be removed at a later UNICOS/mk release.

The `#include` directives will be followed by the usual variable definitions. The structure of a configuration file can now be as follows:

```
#include <cs_predef.h>          /* release 50.0 */
#include <cs_chandef.h>         /* release 50.0 */
#include <cs_iosdef.h>          /* release 50.0 */
#include <cs_tapedef.h>         /* release 50.0 */
#include <cs_diskdef.h>         /* release 50.0 */
#include <cs_netdef.h>          /* release 50.2 */
#include <cs_auditdef.h>        /* release 50.0 */
#include <cs_bootinit_mk.h>     /* release 50.0 */
#include <cs_bootinit.h>        /* release 50.0 */
#include <cs_secdef.h>          /* release 50.0 */
#include <cs_sfsdef.h>          /* release 50.0 */
#include <cs_os_def.h>          /* release 50.1 */
#include <cs_mpp_hwconfig.h>    /* release 50.0 */
```

```
#include <cs_pstdef.h>          /* release 50.0 */


mainframe_t mf[1];

mf[0] {
        ...
}
```

To track and resolve any problems that arise when parsing the configuration file, we recommend that the release number for each header file be appended as a comment to the end of each `#include` directive line (as shown in the preceding example). The release number is given in the first line of each header file. If you are using the install tool, the release number is appended automatically to the newly generated configuration file.

Note: The order in which the include files are specified is extremely important to guarantee that each structure is correctly declared. This order should be the same as the one given in the preceding example.

The brackets (<, >) around each `#include` file instruct the parser to search for each include file in the standard include directory, which is defined as:

```
/opt/CYRIos/1.0/include
```

If your include files are not in this standard directory, you can specify this directory from within the configuration by using the `#incldir` directive at the top of the configuration file, as follows:

```
#incldir </opt/CYRIos/my_include_dir>
```

The above `#incldir` directive instructs the parser to look for each include file in `/opt/CYRIos/my_include_dir`. If it cannot find it there, it will try to locate it under `/opt/CYRIos/1.0/include`.

Although there is no need to, you can also specify the location of each include file relative to the configuration file. You can do this by using the `#include` directive with the name of the file enclosed in double quotes, as follows:

```
#include "../cs_predef.h"       /* release 50.0 */
```

When you use this directive, the parser tries to find the specified file relative to the location of the configuration file you are using to boot your Cray T3E system.

After you have made changes to the configuration file, you should run it through the csp(8) parser to verify that the include files, as well as the variable

definitions, are all correctly given. The following example shows the output of this command.

```
sws% /opt/CYRIos/bin/csp -l log -i param.file
Parsed file "param.file" successfully

sws% tail log
03/03/98 14:53:06 Parsed file "param.file" successfully
03/03/98 14:53:06 csp: normal server start of pid 25594.

03/03/98 14:53:06 csp: Configuration Server stopped.
03/10/98 11:02:26 **** Configuration Server Proxy Started As Pid 3949 ****
03/10/98 11:02:29 Parsed file "param.file" successfully
03/10/98 11:02:29 csp: normal server start of pid 3949.

03/10/98 11:02:29 csp: Configuration Server stopped.
```

Note that a UNICOS/mk system boot will fail if your configuration file contains header files appended at the beginning of the file as well as `#include` directives specifying the header files. The `csp`(8) parser will yield a message indicating that structures are doubly defined.

2.2 New Configuration File Format (UNICOS/mk 2.0)

As of the UNICOS/mk 2.0 release, the configuration server parser, `csp`(8), supports a new format for the configuration file. This new format decreases the size of the configuration file and increases the speed with which the file is parsed.

Note: The older configuration file format is still supported in the UNICOS/mk 2.0.3 release and will be parsed correctly when you boot your Cray T3E system.

With the new format, only one instance of the `mainframe_t` structure needs to be defined, because the rest of the mainframe information is contained in this structure. All other structure declarations and field value assignments can now be specified in nested form inside the `mainframe_t` variable instance.

2.2.1 New Configuration File Format Example

The following is an example of the new configuration file format:

```
mainframe_t  mf[0];

mf[0] {
       os {
            nbuf = 2000;
            sys  = "sn6501";
            node = "galileo";
            ipc {
                  msg {
                        msgmax = 2048;
                        msgmnb = 4096;
                        ...
                      }
                      ...
            }
            panicflush  = 0;
            nc1_min_raw = 20;
            extdcore    = 0;
            ...
       }

       ...

       dev {
             dsk[0] {
                      minor = 1; name = "tmp"; nslices = 1;
                      member[0] {
                          m_type = m_xdd;
                          mstor { phys = &mf[0].dev.xdd[0]; }
                      }
             }

             xdd[0] {
                      minor = 1; name = "tmp";
                      start = 0; length = 12000; length_unit = blocks;
                      pstor = &mf[0].disk_info.phys_stor[0];
             }

             ...
```

```
        }

        ...

} /* mf[0] */
```

In the above example, only the `mainframe_t mf[0]` structure declaration is required. Other structure declarations such as `os`, `dev`, `dsk[0]`, `member` and `mstor` can be defined inside the `mf[0]` instance.

In a similar fashion, structure field values can be specified inside their respective structure declaration. Note that pointer assignments are given before the structure that they point to has been assigned a value (for example, `mstor { phys = &mf[0].dev.xdd[0]; }`).

Value assignments can now be done on a range of array structures. This decreases the size of the configuration file by grouping identical value assignments for array entries into the same declaration, as in the following example:

```
mf[0] {

      ...

      mpp{
               min_cmd = 2;
               min_app = 512;
               sys_status = 0;
               flush_pri = 0;
               pe_actors[0] { /* Command PEs */
                       actor_name[0] = "kernel";
                       actor_name[1] = "PM";
                       actor_name[2] = "fsa";
                       actor_name[3] = "em";
               }
               pe_actors[1] { /* Application PEs */
                       actor_name[0] = "kernel";
                       actor_name[1] = "PM";
                       actor_name[2] = "fsa";
                       actor_name[3] = "em";
               }

           ...
           pe[0:279] {
```

```
                flags = 0; pe_type = PE_TYPE_APP;
                actor_list_index = 0;
        }

        pe[280:307] {
                flags = 0; pe_type = PE_TYPE_CMD;
                actor_list_index = 1;
        }

        pe[308:311] {
                flags = 0; pe_type = PE_TYPE_OS;
                actor_list_index = 2;
        }

        pe[308] { pe_name = "ospe_a"; }

        pe[309] { pe_name = "ospe_b"; }

        pe[310] { pe_name = "ospe_c";
                  actor_list_index = 3; }

        pe[311] { pe_index = 0x137;    /* still needed for boot PE until */
                                       /* SWS 3.6 is installed */
                  pe_name = "ospe_d";
                  actor_list_index = 4; }

    ...
  } /* mpp */
} /* mf[0] */
```

In the above example PEs through 279 will all be application PEs with the same `flags` field value and the same `actor_list_index` list of actors.

Note that OS PEs 308 through 311 are first initialized with an `actor_list_index` value of 2. Later, this value gets rewritten for PEs 310 and 311 to be 3 and 4, respectively. This example shows that fields will always get the last value assignment. Thus, the values of 3 and 4 overwrite the initial `actor_list_index` value of 2 for OS PEs 310 and 311, respectively.

If no values are given to any of the structure fields in the configuration file, they default to 0 if their type is `int_t` or to an empty string if their type is `sstr_t` or `lsstr_t`.

Note: If you have installed the UNICOS/mk 2.0 release or later and the SWS 3.4 release or later you are no longer required to specify the `hz` field in the `pe_t` structure definitions. If a value is given to this field, it will be ignored by the configuration parser.

The same is true for the `pe_index` field with the exception of the boot PE (as shown for OS PE 311 in the preceding example). The `pe_index` field for the boot PE is still required if you are running SWS 3.4 or earlier. This restriction is no longer valid if you are running SWS 3.6 or later.

2.2.2 Formatting the Configuration File

As of the UNICOS/mk 2.0 release, a new `fmtcs`(8) utility is included in the `/opt/CYRIos/bin` directory. The `fmtcs`(8) utility takes a configuration file written in the old style format and rewrites it in the new format style. For a complete description of this tool, refer to the `fmtcs`(8) man page.

To rewrite your configuration file in the new format, run it through the `fmtcs`(8) binary with the `-d` option and provide the name of a new file in which to store this configuration. You need to run `fmtcs`(8) on the SWS workstation, as in the following example:

```
sws% /opt/CYRIos/bin/fmtcs -d param.conf param.new
```

After running `fmtcs`(8), ensure that your configuration file contains the `#include` directives that specify the header files to be read by the configuration file parser, `csp`(8). The configuration file parser supports `#include` directives in UNICOS/mk release 2.0.3 and later. For information on this feature, see “The `#include` Directive”, Section 2.1, page 3.

If you wish, you can append the `cs_`*xxx*`.h` definition files to the top of the `param.new` file instead of using `#include` directives . These files are usually located in the `/opt/CYRIos/1.0/include directory`. The list of files which must be included at the top of the newly formatted configuration file, if you are not using `#include` directives, are as follows:

```
cs_predef.h        cs_chandef.h       cs_iosdef.h       cs_tapedef.h
cs_diskdef.h       cs_netdef.h        cs_auditdef.h     cs_secdef.h
cs_sfsdef.h        cs_os_def.h        cs_pstdef.h       cs_bootinit.h
cs_mpp_hwconfig.h  cs_bootinit_mk.h
```

For information on updating your configuration file to the new format, see “Configuration file changes”, Section 2.3. This section describes the major configuration file changes introduced in the previous and current UNICOS/mk releases.

We recommend that you decrease the number of lines in your param.new file to make it more readable. These changes are not required and the system should boot with what you have, but for aesthetic purposes we recommend the following modifications:

1. Look at any octal and hexadecimal values which were given in the original param file and convert them to their respective representation in the new param file. The reason for this is that the fmtcs(8) utility rewrites all hexadecimal and octal values in decimal when converting between param files.

   These values include: boot_pe, boot_options, io_contr_ppe, cdlimit, and compbits[0] through compbits[8].

2. Rewrite the nw_dev[] entries in the following way:

```
mf[0] {
      ...
      network{
      nw_dev[0] {
               iopath { ioc = 0; iop = 1; chan = 1; }
               ordinal = 0; type = nw_fddi;
      }
      ...
}
```

3. Rewrite the dsk[] entries in the following way:

```
mf[0] {
      ...
      dev {
          dsk[0] {
                    minor = 1; name = "scratch.s40"; nslices =1;
                    member[0] {
                            mtype = m_xdd;
                            mstor { phys = &mf[0].dev.xdd[2]; }
                    }
                    ...
       }
       ...
}
```

4. Rewrite the `phys_stor[]` entries in the following way:

```
mf[0] {
      ...
      disk_info{
            phys_stor[0] {
                      name = @dscsi.s40"; disk_pe = "ospe_a";
                      ptype = d_disk; s_type = d_x500;
                      iopath { ioc = 0; iop = 1; chan = 4; }
                      unit = 0;
             }
             ...
       }
       ...
}
```

5. Rewrite the `pe[]` entries to support array range declarations:

```
mf[0] {
      ...
      mpp{
          pe[0:255] {
                     pe_type = PE_TYPE_APP;
                     actor_list_index = 0;
               }
               ...
       }
       ...
}
```

6. Rewrite the `root`, `dump`, `archive` and `swap` devices:

```
mf[0] {
      ...
      rootdev { name = "root"; minor = 2; }
      swapdev { name = "swap"; minor = 22; }
      dumpdev { name = "dump"; minor = 25; }
      ...
}
```

7. The `fmtcs`(8) binary does not copy any comments from the old `param` file into the new `param` file; you will need to do this manually.

2.3 Configuration File Changes

This section describe fields which are no longer used by the configuration server and fields whose allowed values have changed. Any references to the fields that are no longer used (value assignments) can be removed from the configuration file. For compatibility, these fields still remain part of the structure in which they are defined and should remain so until further notice.

The following configuration file changes are applicable in the UNICOS/mk 1.6.1 release and later:

- All value assignments to the `hz` field in the `pe_t` structure can be removed. Otherwise, they are ignored by the configuration server parser. For compatibility reasons, the `hz` field must not be removed from the `pe_t` structure definition. This will be done at a later UNICOS/mk release.
- All value assignments to the `flush_pri` and `sys_status` fields in the `mpp_t` structure can be removed. Otherwise, they are ignored by the configuration server parser. For compatibility reasons these fields must not be removed from the `mpp_t` structure definition. This will be done at a later UNICOS/mk release.

The following configuration file changes are applicable for the UNICOS/mk 2.0 release and later. These changes also require SWS 3.4 to be installed and running on your system:

- All value assignments to the `pe_index` field in the `pe_t` structure can be removed. Otherwise, theye are ignored by the config server parser. This is true for all `pe_t` structure declarations with the exception of the boot PE.

  The `pe_index` field for the boot PE is still required if you are running SWS 3.4 or earlier. This restriction is lifted if you are running SWS 3.6 or later.

  In the following example, 2 OS PEs are configured and the boot PE has the `pe_index` field defined:

```
pe[6:7] {
         flags = 0; pe_type = PE_TYPE_OS;
         actor_list_index = 2;
}

pe[6] { pe_name = "ospe_b"; }

pe[7] { pe_index = 0x007;      /* still needed for boot PE until */
                               /* SWS 3.6 is installed */
        pe_name = "ospe_d";
        actor_list_index = 4; }
```

- The CS_DEV_MAX macro defined in the cs_diskdef.h definition file has been increased from 256 to 512. This allows sites to have up to 512 xdd slices defined for the Cray T3E mainframe.
- The CS_MAX_DS macro defined in the cs_diskdef.h definition file has been increased from 8 to 32. This allows sites to have up to 32 disk_max_t (diskdriver) structures and possibly to have more than one disk_pe per disk_max_t structure.
- The size field in the audit_t structure (the cs_auditdef.h definition file) has been removed and the int_t panic field has been added. The panic field tells the system whether or not to panic on an audit log write error.
- All value assignments to the num_actors field in the pe_actors structure can be removed; otherwise, they are ignored by the config server parser. For compatibility reasons, the num_actors field MUST NOT be removed from the pe_actors_t structure definition. This will be done for a later UNICOS/mk release.

The following configuration file changes are applicable in the UNICOS/mk 2.0.3 release and later:

- The disable_time field in the security structure can now be set to a value of -1 to indicate that a user will be disabled indefinitely when the maxlogs limit is exceeded. In previous UNICOS/mk releases, only positive values were allowed for int_t fields.
- All value assignments to the user_mls field in the var_t structure can be removed. This field is automatically set to 1 by the UNICOS/mk operating system regardless of the value specified in the configuration file. For compatibility reasons, the user_mls field must not be removed from the var_t structure definition. This will be done at a later UNICOS/mk release.
- All the value assignments to the following fields in the var_t structure can be removed:

nasyn	njobs	nproc	nldmap
nsu_asyn	nexecs	ncrbuf	nsidebuf
nmnt	nldch	ldchcore	
nmt	ntext	ncall	

If you provide value assignments for these fields, they are ignored by the configuration server parser. For compatibility reasons, these fields must not

be removed from the `var_t` structure definition. This will be done for a later UNICOS/mk release.

- The configuration file parser (`/opt/CYRIos/bin/csp`) no longer requires header files to be appended at the beginning of a configuration file. Instead, each header file can be specified through an `#include` directive that instructs the `csp`(8) parser to search and read the specified file. For more information on this feature please refer to "The `#include` Directive", Section 2.1, page 3.

Configuration File Parameters [3]

The UNICOS/mk configuration file on the SWS controls kernel configuration parameters. Because the Cray T3E system is preconfigured, you do not need to make any changes to the UNICOS/mk configuration file in order to boot a usable system. However, you may wish to make configuration changes after your system is running, such as changing the mainframe host name or changing the PE type for a group of PEs. When you modify these parameters, the modifications take effect when you reboot your system.

The default configuration file is `/opt/CYRIos/1.0/config/param.conf`.

You can have multiple configuration files representing different configurations for your system. Before you change the default configuration file, make a copy of the original file under a different name so that it is available if you must restore the default configuration.

This chapter describes some of the parameters in the UNICOS/mk configuration file, and the considerations you must account for when you modify these parameters.

> **Note:** It is strongly recommended that you back up all your configuration changes.

This chapter does not describe the parameters that relate to disk device and file system configuration. For information on I/O controller nodes, see Chapter 8, page 117, and for information on file system configuration, see Chapter 9, page 123.

3.1 Verifying Changes

If you make configuration changes, verify these changes by using the SWS `csv`(8) command before rebooting the system. Use the following format for this command:

```
sws$ /opt/CYRIos/bin/csv -v all -i configfile
```

If no errors are found, you will see output messages similar to the following:

```
Parsed file configuration_file successfully
```

If there are errors in the configuration file, `csv(8)` displays the parameters in question. For more information, see the `csv(8)` man page.

3.2 Host Name Parameter

If you have changed the host name of the mainframe, you should make the same change to the UNICOS/mk configuration parameter `os { node = "`*xxxx*`"; };`. This parameter specifies the system node name; this name should match the host name for the system.

Note: If you change the mainframe host name, you must make this change in **all** of the following places:

- `os { node = "`*xxxx*`"; };` field
- SWS `/etc/hosts` file
- UNICOS/mk `/etc/hosts` file

3.3 Parameters Not to Change

The following parameters are preconfigured based on hardware configuration and should **not** be changed (any changes in the UNICOS/mk configuration file are ignored or overwritten by the SWS configuration database):

```
mf[0] {
      ...

      boot_info {
                  ...
                  archive_size  = xxx;
                  config_size   = xxx;
                  mkpal_size    = xxx;
                  mf_name       = xxx;
                  run_level     = xxx;
                  boot_pe       = xxx;
                  scx_node      = xxx;
      }

      bootstr {
                  dump_mode     = xxx;
                  arch_path     = xxx;
                  dump_path     = xxx;
                  dump_reason   = xxx;
                  sws_hostaname = xxx;
      }
```

```
        mpp_hw {
                cooling_type = xxx;
                phys_npes    = xxx;
                lut_mode     = xxx;
                dpe[*] {
                        pwho       = xxx;
                        attribute  = xxx;
                        links_mask = xxx;
                }
        }

        mpp {
              pe[*] {
                     flags = 0;
                     ...
              }

              ...
        }

        ...
}
```

In addition, the following parameters appear in the default configuration file, but are for PVP systems only. These parameters should **not** be changed on a Cray T3E system. These parameters come from the `cs_pstdef.h` definition file.

- `ncpus` (number of CPUs; PVP systems only)
- `mf_mem_size` (memory size; PVP systems only)
- `mf_chan` (mainframe channels; PVP systems only)
- `ios` (I/O cluster definition; PVP systems only)

3.4 `accept()` Macro Description

The `accept("`*validation_string*`")` macro call is used to define what is an acceptable value for a particular configuration field given in a definition file (`cs_`*xxx*`.h` include file). This macro should be specified right after the field definition. For example, in the `cs_os_def.h` definition file the `var_t` structure has the `nmount` field defined as follows:

```
typedef struct {
        ...

        int_t nmount;
                accept("75 1:100");
        ...
} var_t;
```

This `accept` line tells the configuration server that the `nmount` field has a default value of 75 and a valid range between 1 and 100. If a value outside this range is assigned to this field, then an "out of range" warning message is printed to the `csp`(8) proxy trace file. The value of the field is then set to the default of 75.

If no value is given for the `nmount` field in the configuration file, it defaults to 0. The valid formats for the `accept`() macro are described in the following sections.

3.4.1 Format for Integer Types

For `int_t` (integer) types, the following formats are valid for the `accept`() macro:

`accept("`*number*`")`	Informs the configuration parser that this integer value is the only acceptable value for this field. If a value other than *number* is given to this field, then *number* is assigned as the default value.
`accept("`*number1*`:`*number2*`")`	Informs the configuration parser that the numbers from *number1* to *number2* (inclusive of the boundaries) are acceptable input for this field. If a value outside this range is given, then the lower bound of this range, *number1*, is assigned as the default value.

More than one value or range of values can be used in a single `accept`() directive. Spaces are used to separate these values and default values will be set from the first value or range of values given in the validation string.

Example 1:

```
int_t     nmount;
          accept("0 1 2 3 4 5");
```

This accept() macro informs the configuration parser that any of these values is acceptable for the nmount field. The value of 0 (the first value in the list) is assigned as default if the nmount field is assigned a value which is not 0, 1, 2, 3, 4, or 5.

Example 2:

```
int_t     nmount;
          accept("0:255 512 1024");
```

This accept macro informs the configuration parser that the mount field can be assigned any values between 0 and 255 as well as the 512 or the 1024 value. If a value which is not in this range or is not 512 or 1024 is given to the nmount field, the default value of 0 is assigned to it. This is because the 0:255 range was the first value given in the validation field and is the lower bound in this range.

3.4.2 Format for String Types

For sstr_t and lstr_t (string) types, the following format is valid for the accept() macro:

accept("*string*")	Informs the configuration parser that *string* is the only acceptable value for this field. If a value other than *string* is given to this field, *string* is assigned as the default value.

If no accept() macro is specified for a field of sstr_t or lstr_t type, then any string is a valid value.

3.5 Parameters in Support of Auto-edit

The UNICOS/mk operating system includes the auto-edit feature, which compares the hardware and software configurations when the system is booted. For information on this feature, see Chapter 5, page 69.

To support the auto-edit feature, the following parameters have been added to the configuration file:

min_cmd	The requested minimum number of command PEs.

min_app	The requested minimum number of application PEs.
io_contr_ppe	The physical PE number of an active GigaRing connection.

In addition, there is a new boot options flag that enables the auto-edit feature.

For more information on these configuration file parameters and how they affect the behavior of auto-edit, see Chapter 5, page 69.

3.6 The audit_t Structure

The audit_t structure contains the auditable events. These events make up the system audit event mask. Acceptable values for each event type are 0 = OFF, 1 = ON. These are set in the configuration file by the constants AUDIT_OFF or AUDIT_ON. These constants are defined at the top of the cs_auditdef.h definition file (this file is appended to the top of the configuration file).

A description of each field in the audit_t structure whose value can be set to AUDIT_OFF or AUDIT_ON is shown in the table below. These fields configure the security audit information that gets written into the security log file. The last column shows the value that each field is set to in a default configuration file.

Field	Description	Default value
state	Set auditing state	AUDIT_OFF
all_name	All mkdir, rmdir, link and rm calls	AUDIT_OFF
all_rm	All remove requests	AUDIT_OFF
all_valid	All I/O access requests	AUDIT_OFF
audit	All security auditing criteria changes	AUDIT_ON
chdir	All change directory requests	AUDIT_ON
config	All UDB configuration changes	AUDIT_ON
crl	Cray/REELlibrarian activity	AUDIT_OFF
dac	Discretionary access control changes	AUDIT_ON
discv	Discretionary access violations	AUDIT_ON
filexfr	All file transfer requests	AUDIT_ON

Field	Description	Default value
physio_err	All I/O errors	AUDIT_OFF
ipnet	All IP layer network requests	AUDIT_ON
jend	End-of-Job	AUDIT_ON
jstart	Start-of-Job	AUDIT_ON
linkv	All link (ln) violations	AUDIT_ON
mandv	Mandatory access (MAC) violations	AUDIT_ON
mkdirv	All make directory (mkdir) violations	AUDIT_ON
netcf	Network configuration changes	AUDIT_OFF
netwv	Network violations	AUDIT_ON
nfs	All NFS activity	AUDIT_OFF
nqs	All NQS activity	AUDIT_OFF
nqscf	NQS configuration change	AUDIT_OFF
path_track	File path name access tracking	AUDIT_ON
oper	Operator activities	AUDIT_ON
priv	Use of privilege	AUDIT_OFF
removev	All remove violations	AUDIT_ON
rmdir	All remove directory (rmdir) violations	AUDIT_ON
secsys	All security system call requests	AUDIT_ON
suid	All setuid requests	AUDIT_ON
shutdown	System shutdown requests	AUDIT_ON
startup	System start-up requests	AUDIT_ON
sulog	All su attempts	AUDIT_ON
tape	Tape activities	AUDIT_OFF
user	User name for failed login attempts	AUDIT_OFF
trust	Trusted process activity	AUDIT_OFF
tchg	System time changes activity	AUDIT_ON
panic	Panic system on audit log write error	AUDIT_ON

A default value of AUDIT_OFF is given to those fields whose value is not set in the configuration file.

In addition to the above fields, there are parameters in the audit_t structure that define the security path, maximum size, active security log file name, and retired security log name prefix. These fields are used by the audit log manager server (ALM). A description of these fields follows:

Field	Description
maxsize	Audit log file maximum size. This value should be a whole multiple of the AUDIT_LOG_MAXSIZE (8192000) macro, which is also defined in the cs_auditdef.h definition file.
dir, file	These fields specify the legal directory and file name of where to store the security audit information.
fprefix	This string is prepended to a retired audit log file. When a log file reaches the maxsize limit, it gets retired (renamed) and a new one is generated. The name of the retired log file then becomes fprefix.time_stamp.

3.7 The Flush on Panic Feature

The configuration file contains an entry that determines whether the flush on panic feature is enabled. The parameter appears as follows:

```
panicflush = 1
```

As of the 1.5.1 release of UNICOS/mk, the default value of this parameter is 1, which sets the feature on. In previous releases, the default value was 0, setting the feature to off.

When this feature is enabled, pertinent information is flushed from the file server and disk server when a system panic occurs. This ensures that file systems are not corrupted. On the file server, file buffers are flushed to disk. On the disk server, the partition cache is flushed to disk. If a user has any files open, the files will not be corrupted.

If this feature is not enabled, permanent data in transit when a system panic occurs are not flushed to disk before the system is brought down. This causes

open files to become corrupted and requires you to use the mkfs(8) command to remake file systems when you reboot the system.

3.8 The cs_bootinit.h Parameters

This section describes the definitions and structures of the cs_bootinit.h section of the configuration file, and how these definitions are used in configuring a UNICOS/mk system. You will probably not need to change any of the definitions from their default values.

3.8.1 cs_bootinit.h Definitions

The configuration file contains the following definitions:

Definition	Meaning
CS_MAX_ACTOR_LISTS	The maximum number of UNICOS/mk actor lists that may be defined in any one parameter file.
CS_MAX_ACTORS_PER_PE	The maximum number of UNICOS/mk actors that may be present in the unicosmk.cray-t3e archive (the name implies that it is a per-PE limit, but it is a global system limit).
CS_MAX_PE	The highest logical PE number supported by the config server.
CS_BOOT_INFO_REV_LVL	The revision level of the boot info block, which consists of the parameters in the boot_info_t structure.

PE_TRUTH, PE_LOADED, PE_CONFIGURED, PE_BOOTED, PE_DOWN, PE_DUMP, PE_NOBOOT	The possible flags that may be set in the flags field of the pe_t structure. Many of these flags are for internal UNICOS/mk use; at present, only one of the currently defined flags (PE_NOBOOT) should be specified in the configuration file.

The PE_NOBOOT value is used for each PE in the UNICOS/mk configuration file that has been disabled (with the t3ems(8) utility, in the t3e_config file). For example, to disable LPE 0x21a in the configuration file, the following line in the configuration file would read:

```
pe[538] {
    flags = PE_NOBOOT;
    ...
}
```

3.8.2 The pe_type Enumeration

The pe_type enumeration reads as follows:

```
enum pe_type {
        PE_TYPE_UNDEF,
        PE_TYPE_CMD,
        PE_TYPE_APP,
        PE_TYPE_OS
};
```

These are the possible values that may be used in the pe_type field in the pe_t structure. The PE_TYPE_UNDEF value should not be used in the configuration file; each PE must be defined as one of the three remaining values.

The valid values are PE_TYPE_CMD for a command PE, PE_TYPE_APP for an application PE, and PE_TYPE_OS for an operating system PE. For example, to define LPEs 0x21a through 0x021f to be command PEs, the following line in the configuration file would read:

```
pe[538:543]{
  flags = 0;
  pe_type = PE_TYPE_CMD;
  actor_list_index = 1;
}
```

If a PE is to be disabled (with `flags = PE_NOBOOT`), then we recommend that the `pe_type` value be set to `PE_TYPE_CMD`.

If a PE is a redundant PE, then we recommend that the `pe_type` value be set to `PE_TYPE_CMD`.

3.8.3 The `pe_actors_t` Definition

The `pe_actors_t` definition reads as follows:

```
typedef struct {
    int_t       num_actors;  /* obsolete (to be removed) */
    sstr_t      actor_name[CS_MAX_ACTORS_PER_PE];
} pe_actors_t;
```

This is the definition of the actor list structure in the UNICOS/mk configuration file. For the Cray T3E computer system, there must be at least one instance of this structure. Typically, there are between three and twelve instances of this structure.

Each PE must have an assigned list of actors, which is a subset of the available actors in the `unicosmk.cray-t3e` archive. The `actor_name` field contains the name of each actor.

The `num_actors` field does not need to be specified in the configuration file. It is still required as a field in the `pe_actors_t` structure. This field is ignored by UNICOS/mk releases 2.0 and later and will be removed in a future release.

An OS PE that is specialized for disk I/O might have an actor list that looks like this:

```
pe_actors[7]{
    actor_name[0] = "kernel";
    actor_name[1] = "em";
    actor_name[2] = "PM";
    actor_name[3] = "packet";
    actor_name[4] = "disk";
}
```

The microkernel and some actors (`em` and `PM`) are required on all PEs. The additional actors describe specific OS activities. In this example, the packet server is used to control one or more GigaRing connections, and the disk server is used to control one or more disk peripherals connected to ION devices.

The ordinal in the preceding example, 7, indicates that this is the eighth actor list in the parameter file; the ordinals start at 0.

3.8.4 The pe_t Definition

The pe_t definition reads as follows:

```
typedef struct {
    int_t       reserved;              /* reserved for future use */
    int_t       flags;                 /* PE_NOBOOT, etc. */
    int_t       pe_index;              /* obsolete (to be removed) - LPE */
    enum        pe_type pe_type;       /* PE_TYPE_OS, etc. */
    int_t       actor_list_index;      /* ordinal of actor list for this PE */
    sstr_t      pe_name;               /* name of this PE (OS PEs only) */
    int_t       hz;                    /* obsolete (to be removed) - SYSCLK */
    int_t       in_case_of_fire;       /* reserved for future use */
} pe_t;
```

This is the definition of the per-PE data structure in the UNICOS/mk configuration file. For the Cray T3E computer system, this structure needs to be defined for each physical PE configured in the system.

The fields in the pe_t definition have the following meaning:

Field	Meaning
reserved, in_case_of_fire	Not currently used in UNICOS/mk.
flags	Used to disable one or more PEs at boot time. See the discussion of the PE_NOBOOT flag in Section 3.8.1, page 23.
pe_index	Contains the logical PE number (LPE or LWHO). Ignored by UNICOS/mk release 2.0 or later. Does not need to be specified in the configuration file if you are running UNICOS/mk 2.0 or later and you are running SWS-ION release 3.6 or later. This field will be removed in a future release.
pe_type	Contains one of the values PE_TYPE_CMD, PE_TYPE_APP, or PE_TYPE_OS which translate, respectively, as command PE, application PE, or operating system PE. See the discussion of the pe_type enumeration in Section 3.8.2, page 24.

actor_list_index	Contains the integer ordinal of the appropriate actor list for this PE in the configuration file.
pe_name	Required for operating system PEs (where pe_type = PE_TYPE_OS). It uniquely identifies each OS PE, so that I/O may be directed to the correct OS PE.
hz	Redundant copy of the system clock (SYSCLK) speed, always 75000000 decimal for all Cray T3E computer systems. Ignored by UNICOS/mk release 1.6 or later. Does not need to be specified in the parameter file if you are running UNICOS/mk 1.6 or later and you are running SWS-ION release 3.4 or later. This field will be removed from a future release.

The following is an example of an instance of the pe_t structure in a configuration file. This is logical PE 0x21e, and here it is described as an operating system PE, with the specialized disk I/O configuration in the actor list example in Section 3.8.3, page 25:

```
pe[542]{
    pe_name = "ospe_c";
    flags = 0;
    pe_index = 542;
    pe_type = PE_TYPE_OS;
    actor_list_index = 7;
}
```

By convention, the flags field is specified, even though its value is normally zero. This allows the PE_NOBOOT flag to be added later, if needed.

3.8.5 The mpp_t Definition

The mpp_t definition reads as follows:

```
typedef struct {
    pe_t        pe[CS_MAX_PE];  /* array of PEs */
    pe_actors_t pe_actors[CS_MAX_ACTOR_LISTS];
    int_t       sys_status;     /* Running, Crashing, Dead */
    int_t       flush_pri;      /* Flush-on-panic Priority */
} mpp_t;
```

This is the definition of the MPP-specific data structure in the UNICOS/mk configuration file. For the Cray T3E computer system, one instance of this structure is required.

This structure allocates the space for the maximum number of PEs (the `pe_t` structure) and actor lists (the `pe_actors_t` structure). See the descriptions of those structures in Section 3.8.4, page 26, and Section 3.8.3, page 25.

The `sys_status` and `flush_pri` fields are not currently used.

3.8.6 The `bootstr_info_t` Definition

The `bootstr_info_t` definition reads as follows:

```
typedef struct{
    lstr_t      dump_mode;      /* current dump mode */
    lstr_t      arch_path;      /* path to system archive on SWS */
    lstr_t      dump_path;      /* path to dump of boot PE on SWS */
    lstr_t      dump_reason;    /* reason for the dump */
    sstr_t      sws_hostname;   /* network host name of SWS */
} bootstr_info_t;
```

This data structure is not currently used in UNICOS/mk.

3.9 The `mclock_t` Structure

The `mclock_t` structure is used to configure the central processing unit (CPU) limit parameters for processes and jobs (made of one or more processes) running on a UNICOS/mk system. These parameters are initialized by each process manager server (PM) at system startup.

For information on enforcement of resource limits in UNICOS/mk, see *UNICOS/mk General Administration*.

Field	Description
`clock_major`	The `clock_major` field specifies the number of seconds between checks to see if all processes running on a PE have exceeded their CPU limit. A process's CPU time is calculated by adding its usage of system CPU time, user CPU time, and remote time (time spent processing this process on another PE).

`job_mpp_major`, `job_cmd_major`

Unlike processes, a job's CPU limit is checked in intervals which are multiple of the `clock_major` value. A job's CPU time is calculated by adding the individual CPU times of all its processes.

Job CPU limits are checked every `job_cmd_major` seconds of a `clock_major` interval, for command PEs, and every `job_mpp_major` seconds of a `clock_major` interval for application PEs. Thus, if `job_mpp_major` is set to 300 and `clock_major` is set to 30, the CPU limit for jobs running on application PEs is checked after every 10 intervals (300/30) of a `clock_major` check.

To reduce the interprocess communication (IPC) call overhead to the global process manager (GPM), the `job_cmd_major` value should be larger than the `job_mpp_major` value. This is because there are usually more processes running on a command PE than on an application PE.

The `job_cmd_major` and `job_mpp_major` should be multiples of the `clock_major` value for the above reasons, otherwise, they are effectively rounded to a multiple of it. These values should also be greater than or equal to the `clock_major` value.

`proclim_extra_time`

If a process exceeds its CPU limit, it is not killed immediately. Instead, a`SIGCPULIM` warning signal is sent to this process to allow it to catch this signal and to terminate itself. If after `proclim_extra_time` seconds the process has not terminated, a kill signal is sent to it.

The value for the `proclim_extra_time` field must be a multiple of the `clock_major` value, it it isn't, it is rounded to a multiple of it. This value should also be greater than or equal to the `clock_major` value.

`joblim_extra_time`

Just like processes, if a job exceeds its CPU limit, every process in that job will receive a `SIGCPULIM` warning signal. The value given in

	the joblim_extra_time field is used to give the processes enough time to catch this signal and to terminate themselves before sending them a kill signal. The value for the joblim_extra_time field must be a multiple of the clock_major value; if it isn't, it is effectively rounded to a multiple of it. This value should also be greater than or equal to the clock_major value.
rmttime_action	*Remote time*, a new concept in UNICOS/mk 2.0, is the time spent executing requests on PEs other than the one the process is resident on. Remote time, like system time, tends to be variable depending on system and server activity and is not reproducible between runs of the same command. This configuration variable gives the site the ability to ignore this component during the time limit enforcement check for the process and job. A value of zero means to ignore this value, a value of one means to include this value.
systime_action	*System time* is not a new concept, having been a component of time for all Cray Research systems. It is the time spent in system mode executing requests for a process. This time tends to be variable depending on system activity and is not reproducible between runs of the same command. This configuration variable gives the site the ability to ignore this component during the time limit enforcement check for the process and job. A value of zero means to ignore this value, a value of one means to include this value.
incacct_major	The incacct_major field is not used at this time. This field will be used in the future to specify the period for doing incremental accounting on jobs which have been running over a long period of time. The value for the incacct_major field must be a multiple of the clock_major value; if it is not, it is rounded to

a multiple of it. This value should also be greater than or equal to the `clock_major` value.

The default values and ranges (all in seconds) for the fields in the `mclock_t` structure are shown in the table below.

Field	Default	Range
`clock_major`	30	1 - 300
`proclim_extra_time`	30	1 - 300
`joblim_extra_time`	31	1 - 300
`job_cmd_major`	300	10 - 600
`job_mpp_major`	30	10 - 600
`rmttime_action`	1	0:1
`systime_action`	1	0:1
`incacct_major`	600	300 - 1200

3.10 Stream Buffer Parameters

This section lists the parameters contained in the configuration file that relate to Cray T3E stream buffers. For an explanation of Cray T3E stream buffers, their related parameters, and the values of the parameters, see *UNICOS/mk General Administration*.

Parameter	Description
`BOOT_OPT_STREAMS_DEFAULT_ON`	Used in conjunction with UDB `prohibit-streams` bit to determine results of CLD streams directives.
`BOOT_OPT_STREAMS_OFF`	Unconditionally turns streams off for all applications; provided for testing.

BOOT_OPT_STREAMS_ON	Allows all programs to use streams buffers. Provided for testing; running the system in this mode is not a supported configuration.

3.11 Security Configuration Parameters

This section lists the parameters contained in the configuration file that relate to system security. For a full explanation of the meaning of these parameters and how they relate to security administration, see *UNICOS/mk General Administration*.

3.11.1 Trusted Facility Management

The Trusted Facility parameters in the secure_t structure of the configuration file and their default values are as follows. For more information on these parameters, see *UNICOS/mk General Administration*.

Parameter	Description
priv_su	Determines whether or not the Privilege Assignment List based administrative environment offered by UNICOS/mk will be supplemented with an additional administrative environment that grants complete administrative authority to processes belonging to the root user. The default setting is 1. The required Cray ML-Safe setting is 0.
catbits	Array of integer values specifying the bit positions in the low order 18 bits (the portion masked by the octal value 0777777) of a word that represent site defined administrative categories. Bits above the low order 18 bits are reserved for use by Cray Research and should not be used to define site-defined administrative roles. No user defined categories are defined by default. This configuration item has no required Cray ML-Safe setting.

`category`	Array of names containing the set of site-defined administrative category names that will map to bit positions in the list of site-defined administrative categories (as described in `security.catbits`). No user defined categories are defined by default. There is no required Cray ML-Safe setting.

3.11.2 Discretionary Access Management

This configuration file contains one parameter that defines discretionary access control. For more information, see *UNICOS/mk General Administration*.

Parameter	Description
`fsetid_restrict`	Determines whether the owner of a file can make that file a set-UID or set-GID file without additional authorization. The default setting is 0. There is no required Cray ML-Safe setting.

3.11.3 Mandatory Access Control

This section describes the mandatory access control (MAC) parameters in the configuration file. For more information on these parameters, see *UNICOS/mk General Administration*.

Parameter	Description
`compbits`	Array containing the definitions of compartment bit positions used to compose MAC labels. Although systems come with certain compartments defined in the parameter file by default, sites may redefine these compartments to suit their own needs. There is no required Cray ML-Safe setting.
`comparts`	Array containing the names of each compartment described in the `security.compbits` array. There is no required Cray ML-Safe setting.

`lvlnums`	Array containing the values (ranging from 0 to 16) to be used as the level components of MAC labels. Systems are shipped with this array initialized to settings of 0 to 16 respectively for each element of the array. Generally there is no reason to change this default setting, but sites may do so if they choose. There is no required Cray ML-Safe setting.
`lvlname`	Array defining the names to which the levels defined in `security.lvlnums` map. Although systems are shipped with a simple set of names defined for the level names, sites that use levels for protection are encouraged to define their own level names. There is no required Cray ML-Safe setting.
`minslevel`	Numeric value of the level component of the minimum MAC label at which processing is allowed on the system. The default setting is 0. There is no required Cray ML-Safe setting.
`maxslevel`	Numeric value of the level component of the maximum MAC label at which processing is allowed on the system. The default setting is 0. There is no required Cray ML-Safe setting.
`sysvcomps`	Numeric representation of the set of compartments allowed in the maximum MAC label at which processing is allowed on the system. The default setting is 0. There is no required Cray ML-Safe setting.

`secure_mac`	Flag indicating whether you have set up all the necessary conditions to use system high and system low labels. The default setting is 0. The required Cray ML-Safe setting is 1.
`dev_enforce_on`	Flag indicating whether to enforce strict device labeling rules. The default setting is 0. The required Cray ML-Safe setting is 1.
`forced_socket`	Flag indicating whether to require `syslogd`(8) to use a socket instead of a named pipe to communicate with its clients. The default setting is 0. The required Cray ML-Safe setting is 1.
`secure_pipe`	Flag indicating whether to require MAC write access to a pipe in order to grant read access to the pipe. The default setting is 0. The required Cray ML-Safe setting is 1.

3.11.4 UNICOS/mk Identification and Authentication (I&A)

This section describes the UNICOS/mk Identification and Authentication (I&A) parameters in the configuration file. For more information on these parameters, see *UNICOS/mk General Administration*.

Parameter	Description
`random_pass_on`	Flag indicating whether to require users to use machine-generated passwords. The default setting is 0. There is no required Cray ML-Safe setting.
`pass_minsize`	Specifies the minimum length of a machine-generated password. The default setting is 8. There is no required Cray ML-Safe setting.

`pass_maxsize`	Specifies the maximum length of a machine-generated password. The default setting is 8. There is no required Cray ML-Safe setting.
`disable_acct`	Flag indicating whether a user's account should be disabled if the user has too many consecutive failed login attempts. The default setting is 0. There is no required Cray ML-Safe setting.
`disable_time`	Specifies the length of time in seconds that a user's account will be disabled after the maximum number of consecutive failed login attempts has been exceeded. As of the UNICOS/mk 2.0.3 release, this field can be set to a value of –1 to indicate that a user's account will be disabled indefinitely when the `maxlogs` limit is exceeded. In previous UNICOS/mk releases, positive values only were allowed for this field. The default setting is –1. There is no required Cray ML-Safe setting.
`maxlogs`	Specifies the number of consecutive failed login attempts allowed for a given user before the account becomes disabled. The default setting is 5. There is no required Cray ML-Safe setting.
`console_msg`	Flag indicating whether to post a message to the system console when a user exceeds the number of consecutive failed login attempts specified in `security.maxlogs`. The default setting is 0. There is no required Cray ML-Safe setting.

`logdelay`	Specifies the number of seconds that will elapse between login prompts for a failed login attempt. The default setting is 0. There is no required Cray ML-Safe setting.
`delay_mult`	Specifies a multiplier to be applied to the delay between failed login attempts for each successive attempt. The default setting is 0. There is no required Cray ML-Safe setting.

3.11.5 Object Reuse

This section describes the parameters in the configuration file related to object reuse. For more information on these parameters see *UNICOS/mk General Administration*.

Parameter	Description
`declassify_disk`	Flag indicating whether the `spclr -d` option is enabled at your site. The default setting is 0. There is no required Cray ML-Safe setting.
`declassify_pattern`	Specifies the pattern that will be used by `spclr -d` to begin its sequence of overwriting disk blocks. The default setting is 0. There is no required Cray ML-Safe setting.
`overwrite_count`	Specifies the number of passes `spclr -d` will make when overwriting disk blocks. The default setting is 3. There is no required Cray ML-Safe setting.
`sanitize_pattern`	Specifies the pattern that will be used by `spclr -s` when overwriting a file. The default setting is 0. There is no required Cray ML-Safe setting.

`secure_scrub`	Flag indicating whether to automatically overwrite the block in a file with the pattern specified in `security.sanitize_pattern` when a file is removed. Significant performance degradation can occur if this flag is enabled. The default setting is 0. There is no required Cray ML-Safe setting.

3.11.6 Miscellaneous Security Settings

This section describes miscellaneous security parameters in the configuration file. For more information on these parameters see *UNICOS/mk General Administration*.

Parameter	Description
`nfs_secure_export_ok`	Flag indicating whether information on a secure file system can be exported over Network file system (NFS). The default setting is 1. There is no required Cray ML-Safe setting.
`nfs_remote_rw_ok`	Flag indicating whether remotely mounted NFS file systems can be mounted as read/write or whether they must always be read-only files. The default setting is 1. There is no required Cray ML-Safe setting.
`secure_net_options`	Bit mask of flags that may contain the following bits:

Bit name	Value	Description
NETW_STRICT_B1	00001	Restrict network label ranges so that network access list (NAL) entries must either specify an Internet Protocol (IP) security option or a single label for a remote host.
NETW_SOCK_COMPAT	00002	Automatically make all sockets set up by a privileged process multi-level sockets.
NETW_RCMD_COMPAT	00004	Allow traditional processing of .rhosts and hosts.equiv.

The default setting item for secure_net_options is 6 (NETW_SOCK_COMPAT | NETW_RCMD_COMPAT). The 1 (NETW_STRICT_B1) bit is required for an Cray ML-Safe configuration.

secure_system_console — Obsolete.

system_admin_console — Obsolete. Must always be set to /dev/console.

secure_operator_console — Obsolete. Must always be set to /dev/console.

Subsystem Configuration Tool [4]

The ConfigTool utility lets you configure the system daemons and the `/etc/config` configuration files using a graphical user interface. ConfigTool lets you select the particular subsystems to be configured, and in turn invokes the appropriate utility (`inmenu` or an asynchronous configurator) to perform the actual editing of subsystem configuration files. You will use ConfigTool after the Cray T3E mainframe has been booted.

Note: ConfigTool does not deal with any configuration parameters that are part of the configuration parameter file; the `pact`(8) editor that runs on the system workstation (SWS) handles the configuration of the configuration file.

ConfigTool lets you do the following:

- Directly import existing subsystem configuration files
- Edit multiple subsystems concurrently
- Review changes before committing them
- Save configuration files under different names

You can also edit subsystem configuration files manually using an editor such as `vi`(1). When you later invoke ConfigTool, the changes you made with `vi` will appear because ConfigTool directly imports subsystem configuration files, rather than maintaining a separate database.

4.1 Invoking ConfigTool

To invoke ConfigTool, enter the `configtool`(8) command in a UNICOS/mk window:

```
configtool [-b] [-c config_mount_dir]
```

`-b`

Backs up the original or default subsystem configuration file as it exists before overwriting it with changes. By default, backup copies are made, but this may be changed with resources.

`-c` *config_mount_dir*

Specifies the default directory mount point from which subsystem configuration files will be loaded and into which they will be saved. Subsystem configuration files are expected to be in specific directories. You can use the `-c` option to specify a different top-level location. The *config_mount_dir* value is the top-level (or mount-point equivalent). For example, the path to the `acct_config` subsystem configuration file is expected to be `/etc/config/acct_config`. If you wanted to place the file in `/home/system/config/etc/config/acct_config`, you would specify `/home/system/config` for the `-c` value.

Note: The menu system program (`inmenu`) is a single binary that implements both the X Window System and `curses` versions of the menu system. Because of this, even if you select the `curses` version of the menu system, `inmenu` will try to contact `Xlib` and, if your `xserver` is disabled, you will receive an error message (which you can ignore) and then enter the `curses` version of the menu system. If your `xserver` is disabled with the goal of having secure X Window System access, we recommend that you use `xauth` to ensure that the X packets are secure between the mainframe and the workstation.

Note: The `inmenu` program cannot display the GUI interface on an X Window System version that is running with either the `TrueColor` or `DirectColor` visuals.

Figure 1 shows the main window.

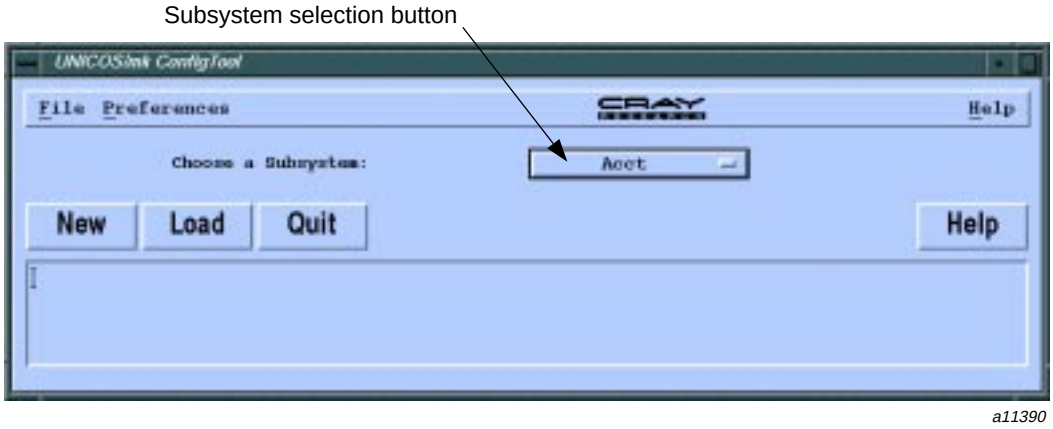


Figure 1. ConfigTool Window

4.2 Using the Menus and Buttons

The ConfigTool window has the following features:

- A menu bar with three pulldown menus:
  - `File`, which allows you to create a new subsystem configuration file or load existing subsystem configuration files.
  - `Preferences`, which allows you to set preferences for working with ConfigTool and `inmenu`.
  - `Help`, which provides instructions for using ConfigTool.
- Buttons that make the `File->New`, `File->Load`, `File->Quit` and `Help->Overview` functions readily available.
- A button that allows you to view and select the subsystems modified by ConfigTool.
- A message window that displays information about the selected subsystem. Error messages appear in pop-up windows.

Table 1 summarizes the menu options.

Table 1. ConfigTool Menus

Menu	Submenu	Description
File	New	Creates a new default configuration file for the chosen subsystem and starts the inmenu interface.
	Load	Loads an existing configuration file for the chosen subsystem and starts the inmenu interface. You will be prompted for a file name.
	Quit	Quits the ConfigTool session.
Preferences	Configuration directory	Specifies the directory into which configuration files will be saved by default.
	Back up old configuration file	Toggles the ability to automatically save a backup version of the original or default subsystem configuration file as it exists before overwriting it with your changes.
	Escape to Shell	Creates a new window with the shell specified in the inmenu preferences.
	Preferences for editing subsystem	Invokes a special version of inmenu in which you can change its defaults. Things that may be changed include the following: • Editor • Shell • Pager • Input editor keys • Permissions on menu files (those that end in .mnu) • Scroll bar Changes that you make in this edit session will take affect for all subsequent inmenu edit sessions.
Help	Overview	Provides an introduction to ConfigTool. This is the help text that is also available from the Help button.
	Index	Provides an index to the content of the various help files.

Menu	Submenu	Description
	`Table of Contents`	Lists the help topics.
	`Tasks`	Tells you how to perform the following tasks: • Select a subsystem • Load an existing subsystem configuration file • Create a new subsystem configuration file • Review changes you have made to a subsystem configuration file • Verify changes you have made to a subsystem configuration file • Save a subsystem configuration file • Exit from a subsystem edit session • Exit from ConfigTool • Modify the working environment for ConfigTool • Modify the working environment for `inmenu` • Modify ConfigTool startup environment
	`References`	Lists related files and documentation.
	`On Item`	Provides help for the selected ConfigTool item. Selecting `On Item` changes the pointer to a question mark with an arrow at the bottom. You can then point to any of the ConfigTool buttons to find the help for that particular subject.
	`Using Help`	Provides help about the help tool.
	`Application`	Provides the released version number, comment contact, and copyright statement.

4.3 Getting Help

You can get help for using ConfigTool by selecting a topic from the `Help` menu. For details about the `Help` menu selections, see Table 1, page 44. The `Help` button on the ConfigTool window provides ready access to the `Help->Overview` text. In addition, a `Help` button is provided on pop-up windows. The `inmenu` and asynchronous configurator programs also provide help.

4.4 Selecting a Subsystem

To select a subsystem, do the following:

1. Move the mouse pointer over the button next to `Choose a Subsystem:`
2. Press and hold the left mouse button to see the list of subsystems
3. Point to the correct subsystem name and release the mouse button

After you have selected a subsystem, the window below the buttons will be updated with information about the subsystem.

Figure 2 shows the selection of the `Nfs-export` button, which allows you to edit the configuration of local file systems that are exported via NFS.

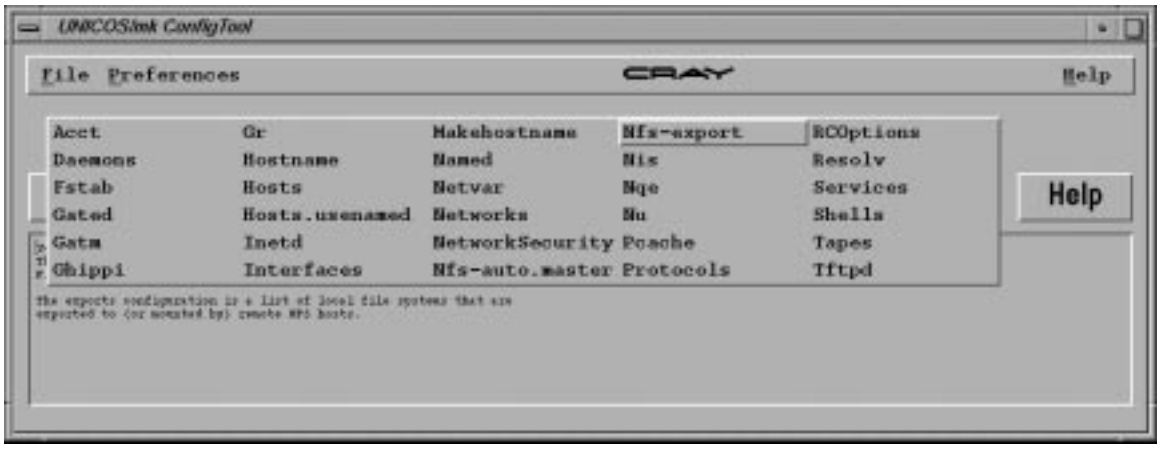


Figure 2. Selecting a Subsystem

Table 2 lists the subsystems that may be configured through ConfigTool.

Table 2. ConfigTool Subsystems

Button	Default file	Description
Acct	/etc/config/acct_config	Selects the accounting subsystem. For more information, see the acct(5) and acct(8) man pages.
Daemons	/etc/config/daemons	Selects the system daemons subsystem, which lets you configure daemons that are started at system startup. For more information, see the sdaemon(8) man page.
Fstab	/etc/config/fstab	Selects the file system configuration subsystem, which lets you configure the relationship of file systems to the logical devices and directory structures in the system. For more information, see the fstab(5) man page.
Gated	/etc/gated.conf	Selects the gate daemon subsystem, which lets you configure the routing of TCP/IP packets on the mainframe. For more information, see the gated-config(5) man page.
Gatm	/etc/config/atm.pvc	Selects the GigaRing asynchronous transfer mode (ATM) interfaces subsystem, which lets you configure the remote hosts that this system will communicate with by way of the GigaRing ATM interfaces. For more information, see the atmarp(8) man page.
Ghippi	/etc/ghippi*.arp	Selects the GigaRing Hippi address resolution protocol subsystem, which lets you configure the GigaRing Hippi hardware addresses to remote machines. For more information, see the hippi(4) man page.

Button	Default file	Description
Gr	/etc/gr*.arp	Selects the GigRing host-to-host address resolution protocol subsystem, which lets you configure the GigaRing host-to-host hardware address to remote machines. For more information, see the arp(4P) man page.
Hostname	/etc/config/hostname.txt	Selects the host name subsystem, which lets you configure the host name of the mainframe.
Hosts	/etc/hosts	Selects the host address lookup subsystem, which lets you configure the method by which applications will look up hosts and addresses. For more information , see the hosts(5) man page.
Hosts.usenamed	/etc/hosts.usenamed	Selects the Hosts.usenamed subsystem, which lets you turn on domain name service. For more information, see the hosts(5) man page.
Inetd	/etc/inetd.conf	Selects the generic internet daemon configuration subsystem, which lets you configure the inetd daemon for a variety of internet services provided by other daemons. For more information, see the inetd(8) man page.
Interfaces	/etc/config/interfaces	Selects the network interface configuration subsystem, which lets you configure the network interfaces for the mainframe. For more information, see the ifconfig(8) and initif(8) man pages.
Makehostname	/etc/config/makehostname	Selects the host name subsystem, which lets you configure the host name of the mainframe.
Named	/etc/named.boot	Selects the domain name server subsystem, which lets you configure the named domain name server process daemon. For more information, see the named.boot(5) man page.

Button	Default file	Description
Netvar	/etc/config/netvar	Selects the network variables subsystem, which lets you specify the boot-time arguments to /etc/netvar. For more information, see the netvar(8) man page.
Networks	/etc/networks	Selects the network address configuration subsystem, which allows you to configure the database of all known addresses on the Internet. For more information, see the networks(5) man page.
NetworkSecurity	/etc/config/spnet.conf	Selects the network security subsystem, which lets you configure security for various network activities. For more information, see the spnet(8) man page.
Nfs-auto.master	/etc/auto/auto.master	Selects the network file system (NFS) automaster and automaps subsystem, which lets you configure the maps used for automounting. For more information, see the automount(8) man page.
Nfs-export	/etc/exports	Selects the NFS exported file system subsystem, which lets you configure the list of local file systems that are exported to (or mounted by) remote NFS hosts. For more information, see the exports(5) man page.
Nis	/etc/config/ypdomain.txt	Selects the network information service subsystem, which lets you configure the network information service (formerly known as yellow pages, or YP).
Nqe	/etc/nqeinfo	Selects the network queueing environment (NQE) subsystem, which lets you modify the NQE configuration. NQE uses an asynchronous configurator rather than inmenu. For more information, see the nqeinit(8) man page.

Button	Default file	Description
Nu	/etc/nu.cf60	Selects the new user subsystem, which lets you set up various defaults that the nu(8) command will use when creating, modifying, and deleting user accounts. For more information, see the nu(8) man page.
Pcache	/etc/config/pchlist	Selects the partition cache subsystem, which lets you configure the partition cace associated with a device. For more information, see the pcache(8) man page.
Protocols	/etc/protocols	Selects the protocols subsystem, which lets you edit the /etc/protocols file. For more information, see the protocols(5) man page.
RCOptions	/etc/config/rcoptions	Selects the resource environment subsystem, which lets you configure the environment variables used to control the execution of the /etc/rc file at system startup. For more information, see the brc(8) man page.
Resolv	/etc/resolv.conf	Selects the domain name server resolution subsystem, which lets you configure the method by which the local system will contact name servers in order to look up host names and addresses. For more information, see the resolv.confg(5) man page.
Services	/etc/services	Selects the networking services subsystem, which lets you configure the list of networking services known to the UNICOS/mk operating system. For more information, see the services(5) man page.
Shells	/etc/shells	Selects the user shells subsystem, which lets you specify the list of valid user shells. For more information, see the shells(5) man page.

Button	Default file	Description
Tapes	/etc/config/text_tapeconfig	Selects the tape configuration subsystem, which lets you configure the tapes used with the UNICOS/mk operating system. For more information, see the text_tapeconfig(5) man page.
Tftpd	/etc/tftpd.conf	Selects the tftp daemon subsystem, which lets you configure the list of directories that are accessible to requests made to the TFTP daemon and restrict read and write access to the contents of the directories. For more information, see the tftpd(8) man page.

4.5 Creating a New Subsystem Configuration File

To create a new subsystem configuration file, do the following:

1. Select the correct subsystem. See Section 4.4.
2. Click the New button. This is the same as selecting File->New.

ConfigTool will then open an editing session that contains a default configuration file for the subsystem. Form files will be empty. Only menu selections will have default values. You can then edit and save your changes. Figure 3 shows an example of the inmenu Menu Viewer editing session for the network interfaces subsystem.

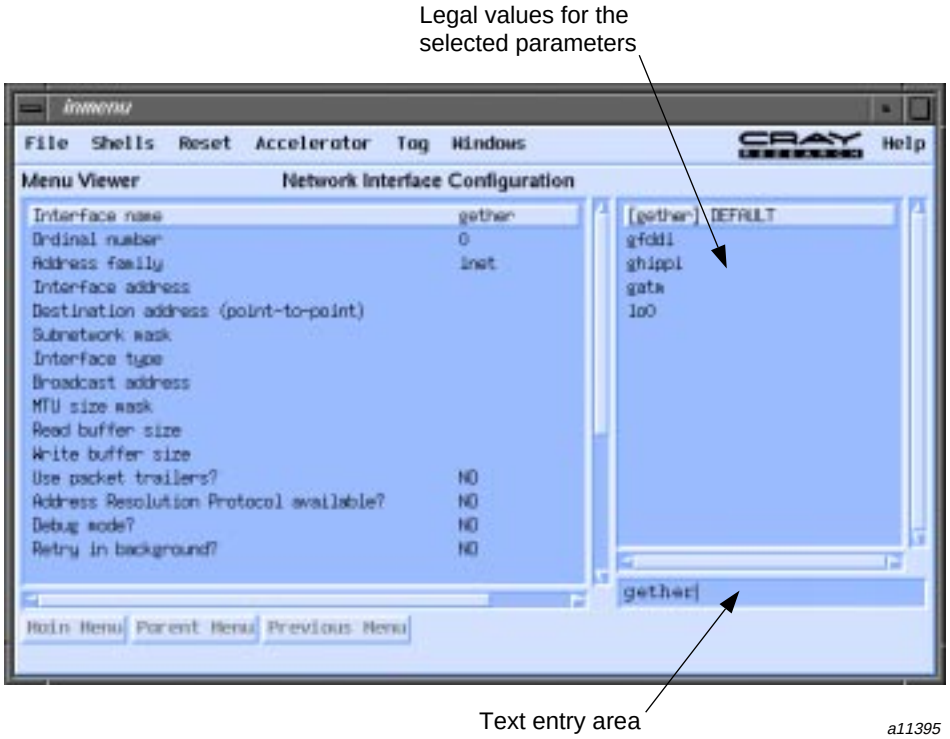


Figure 3. Menu Viewer

Caution: If you do not save your changes before exiting the editing session, they will be lost.

For information about using `inmenu`, see Section 4.15, page 57.

4.6 Loading an Existing Subsystem Configuration File

To load an existing subsystem configuration file, do the following:

1. Select the correct subsystem. See Section 4.4.
2. Click the `Load` button. This is the same as selecting `File->Load`.

The pop-up window shown in Figure 4 will prompt you for the name of the subsystem configuration file; the default name is provided in the pop-up window in the `Selection` field. (The default name is dependent upon the subsystem chosen.) If you wish to load a file under a different name, change the pop-up window's `Filter` text field to contain the correct path.

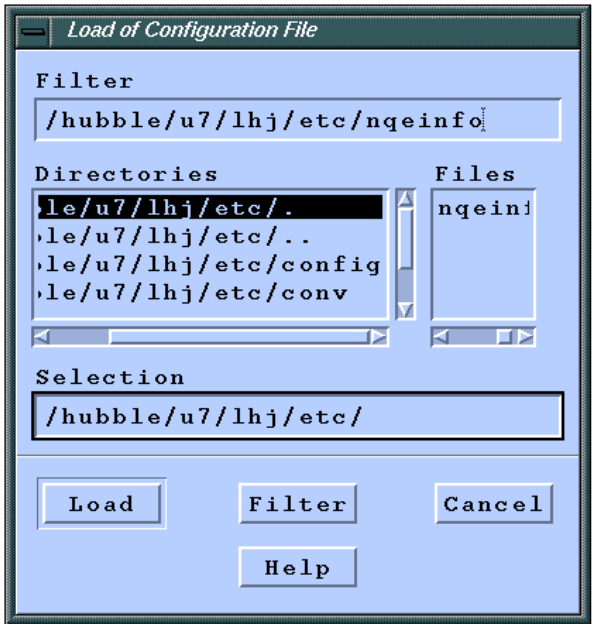


Figure 4. File Selection Box

Note: The path of your saved file may differ from the default. If so, change the last part of the filter field to an expression that matches the file name you want. After changing the filter field, click on the `Filter` button at the bottom of the pop-up window to produce a list of matching files.

After the file appears in the `Files` area, you can select it by doing one of the following:

- Double-click on the file name
- Enter the file name directly into the `Selection` text field and click the `Load` button

ConfigTool will then make a copy of the file in a temporary directory. It will then start an edit session in a new window for that subsystem configuration file. You can then edit and save your changes.

Caution: If you do not save your changes before exiting the `inmenu` edit session, they will be lost.

For information about using `inmenu`, see Section 4.15, page 57.

4.7 Reviewing Your Changes

To compare the current version of your file with the default version (when creating a new file) or original version (when loading an existing file), select `File->Review` from the `Menu Viewer` screen of `inmenu`. This process may take a few minutes. ConfigTool will then use the `sdiff`(1) command to produce a side-by-side differences window that shows the differences between the original / default file and the configuration file you are editing. This will allow you to see additions, deletions, and changes between the two versions.

For more information about using `inmenu`, see Section 4.15, page 57.

4.8 Verifying Your Changes

You can perform basic verification such as syntax checking by using the `File->Verify` menu selection.

Note: This feature is not implemented for all the subsystems.

For more information about using `inmenu`, see Section 4.15, page 57.

4.9 Saving Your Changes

To save your changes, select `File->Save` from the `inmenu` Menu Viewer. It may take some time to create an updated version of the subsystem configuration file in memory. After this has been completed, ConfigTool will prompt you to provide a directory and file name. After providing the name, click on the `Save` button or press the return key in the `Selection` text field to proceed.

If you have turned on the `Backup old configuration file` option in the `Preferences` menu, the original or default subsystem configuration file will be saved before the changed or new configuration file is written. The file name will be *filename*`.bak`.

For more information about using `inmenu`, see Section 4.15, page 57.

4.10 Exiting from `inmenu`

To leave the `inmenu` editing session, select `File->Quit` from either the `inmenu Menu Viewer` window or the `Form Viewer` window.

Caution: If you do not save your changes before exiting the editing session, they will be lost.

4.11 Exiting from ConfigTool

You can exit ConfigTool and exit all editing sessions selecting the `Quit` button in the ConfigTool main window menu bar.

ConfigTool will then provide a confirmation prompt. If you click the `OK` button, ConfigTool will kill all subsystem editing sessions (losing any changes that were not saved) and will then exit.

4.12 Modifying the ConfigTool Working Environment

You can change the ConfigTool working environment while it is running by using the following menu selections:

- `Preferences ->Configuration directory`

  This selection lets you specify the default directory mount point from which subsystem configuration files will be loaded and into which they will be saved. Subsystem configuration files are expected to be in specific directories. You can use this selection to specify a different top-level location. The value you specify is the top-level (or mount-point equivalent). For example, the path to the `acct_config` subsystem configuration file is expected to be `/etc/config/acct_config`. If you wanted to place the file in `/home/system/config/etc/config/acct_config`, you would specify `/home/system/config`.

- `Preferences ->Backup old configuration file`

  This selection lets you toggle the ability to automatically save a backup version of the original or default configuration file as it exists before overwriting it with your changes. By default, backup copies are made.

Note: The preference changes for the directory and backup capability only apply to the current ConfigTool session. To make these changes for future invocations of ConfigTool, see Section 4.14, page 56.

4.13 Modifying the inmenu Working Environment

If you wish to change the environment for your inmenu editing session, use the following menu selection:

```
Preferences
    ->Preferences for editing subsystem
```

This will bring up a special version of inmenu in which you can change its defaults. Elements that may be changed include the following:

- Editor
- Shell
- Pager
- Input editor keys
- Permissions on menu files (those that end in .mnu)
- Scroll bar

4.14 Modifying the ConfigTool Startup Environment

ConfigTool has several user-configurable resources to maximize the way you want the tool to respond and provide information to the user. ConfigTool provides a command line method to turn on or change these resources.

ConfigTool searches for resources in the following files, in order:

1. ./.configtoolrc
2. $HOME/.configtoolrc

Therefore, resources defined in $HOME/.configtoolrc will override those defined in ./.configtoolrc. Command line arguments will override these resources.

Table 3 shows the resources that may be changed.

Table 3. ConfigTool Resources

`.conftoolrc` resource	Description
`Backup_Button=ON|OFF`	Sets the default for the backup feature. When set to `ON`, a backup copy of the original or default subsystem configuration file will be made before current changes are saved. The backup copy is named *filename*`.bak`. You can also turn on the back up feature by using the `configtool -b` option. The default is `ON`.
`config_dir=`*config_mount_dir*	Specifies the directory from which subsystem configuration files are loaded and saved. If not the root directory, this directory is the top-level or mount-point equivalent. (For example, the mount-point equivalent of `/home/system/config/etc/config/acct_config` is `/home/system/config`). You can also specify this directory by using the `configtool -c` option. The default for *config_mount_dir* is `/home/system/config`.

4.15 `inmenu` Reference

You will use the `inmenu` utility to perform the actual editing of most subsystem configuration files. (Some subsystems have their own configurator.) The following sections describe the details of using `inmenu`.

4.15.1 Menu Screen

Figure 5 shows an example of the `inmenu` main window.

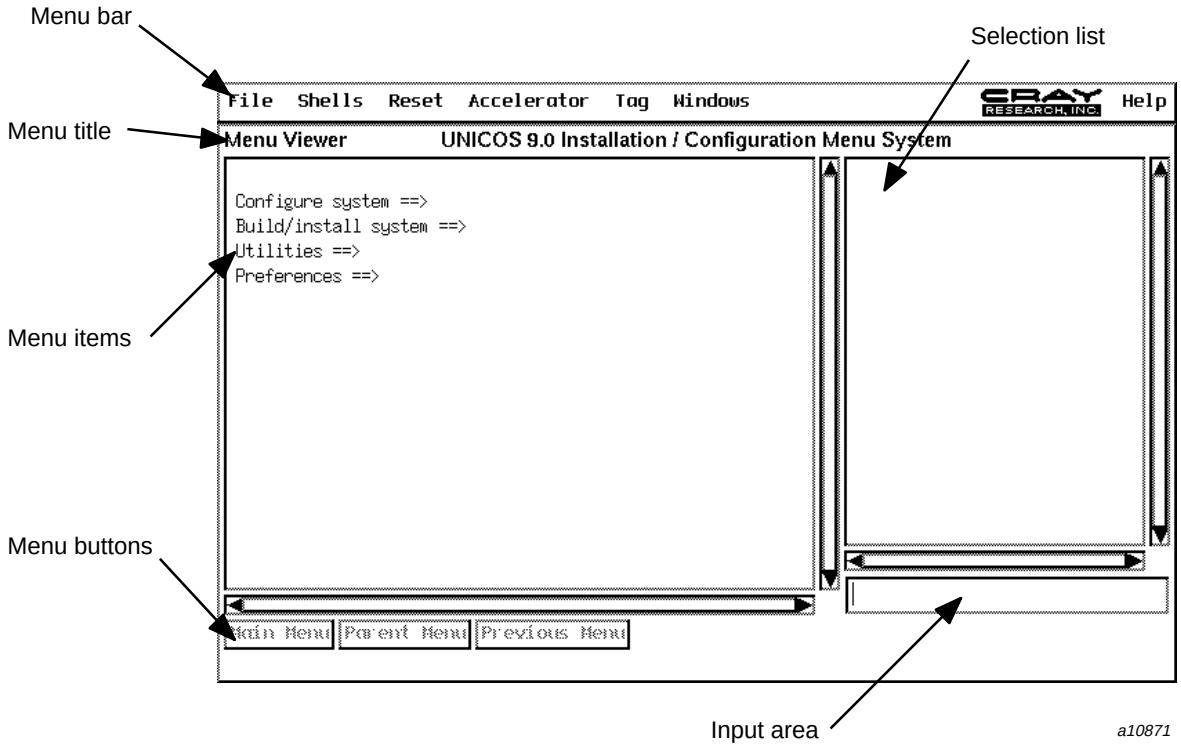


Figure 5. Main menu window

4.15.2 Menu Items

As shown in Figure 6, the menu title area displays the name of the current menu. The menu items window area displays menu items that can be selected. The current menu item selected is highlighted. Select another menu item by clicking on that item. The following types of menu items can be selected:

- An *action* executes a program. An action is followed by three dots (. . .). It initiates the action specified in a separate window and prevents you from doing other operations until the action completes.
- A *selection* is a configuration parameter with an associated value. It displays a list of valid selection values in the selection list window to the right of the menu items window. You can change the current value by clicking on any of the selection values shown or by entering the value in the input area located under the selection list.

- A *menu* leads to another menu, one level down in the menu tree structure. A menu is designated by ==>. Use the Parent button to return to the previous menu level.
- A *comment* is informational only; it does nothing.

4.15.3 Menu Bar

This window area contains menu system pull-down menus. To make a selection from the menu bar, click on an item to display the associated pull-down menu. Then click on the desired option from the menu. To make the pull-down menu disappear, just click on the menu bar. Available pull-down menus are as follows.

4.15.3.1 File Menu

The File pull-down menu contains the following option:

Exit <CTRL-Q>	Quits the menu system and prompts you to update changes applied to the current form menu, if any, before exiting.

4.15.3.2 Shells Menu

The Shells pull-down menu contains the following options:

Ksh	Creates a separate window running ksh (the standard shell).
Sh	Creates a separate window running sh (which should be ksh).
Csh	Creates a separate window running a C shell.
Preference shell	Creates a separate window running a shell defined by the Preferences menu option Type of Shell. The default is the standard shell.

4.15.3.3 Reset Menu

The Reset pull-down menu contains the following options:

Reset selections	Resets all selection values in the current menu (basically the same as an undo command).
Undo effects of last reset	Undoes the effects of the last Reset selections command. All selections that

changed after the Reset command was executed are lost when this option is selected. This option restores selection values to the values they had before the Reset command was executed.

4.15.3.4 Accelerator Menu

The Accelerator pull-down menu contains the following options:

Assign accelerator key	Lets you assign current menu items to one of the keys 1 through *n*, where *n* is defined by the Preferences menu selection Maximum number of accelerator keys. With this option, you can define time-saving shortcuts for moving around the menu tree.
Select accelerator key	Displays the list of accelerator keys you have already defined. If desired, you can select one to traverse the menu system to the menu defined by the chosen key.
Next accelerator key	Selects the next accelerator key in the displayed list and goes to the specified menu.
Previous accelerator key	Selects the previous accelerator key in the displayed list and goes to the specified menu.

4.15.3.5 Tag Menu

The Tag pull-down menu contains the following options:

Find selection tag	Searches the menu tree for the specified search string. If one or more matches for the string are found, this option traverses the menu system to get to the menu defined by the first match found. The maximum number of matches saved is defined by the Preferences menu selection Maximum number of search keys. This option is useful if you know what parameter you

	want to change, but do not know the menu on which it appears.
`Select selection tag`	Displays a list of all defined tags and lets you select one if desired.
`Next selection tag`	Finds the menu containing the selection defined by the next selection tag.
`Previous selection tag`	Finds the menu containing the selection defined by the previous selection tag.

4.15.3.6 Windows Menu

The Windows pull-down menu contains the `Tree Viewer` and `Form Viewer` options.

4.15.3.6.1 Tree Viewer

The `Tree Viewer` option displays or hides the menu tree. The menu tree shows all of the currently accessible menus.

Besides the menu tree, the `Tree Viewer` window contains a panner window and a menu bar. The panner window quickly moves you to another part of the tree. The box in the panner window matches what is shown in the tree. Just move the panner window to another part of the tree to have it displayed in the `Tree Viewer` window.

You can go to any menu in the tree by clicking on the desired button. Clicking the left mouse button on the background of the tree and dragging the background also moves the view of the tree in the direction you drag the mouse.

The menu bar contains `Tree Viewer` pull-down menus. The only `Tree Viewer` menu option that is not described elsewhere in this chapter is the `Menu Viewer` option, which displays or hides the main menu and lets you edit a specific object.

4.15.3.6.2 Form Viewer

The `Form Viewer` option displays or hides the form viewer. A form or *form menu* is a screen representation of the record lines that make up a menu system configuration file. See Figure 6, page 62, for a sample configuration file, and Figure 7, page 63, for a sample form menu.

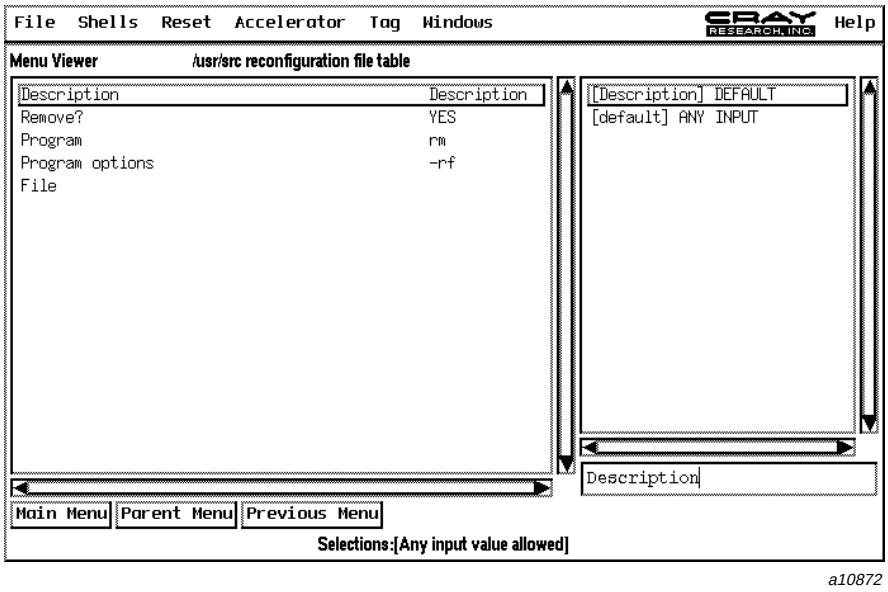


Figure 6. Sample configuration file

The `Form Viewer` option displays or hides multiple objects that are the same type; that is, it displays a list of objects that have the same editable fields (in contrast to the `Menu Viewer`, which lets you edit a specific object). For example, the `Form Viewer` may display a list of disk slices. Clicking on a specific disk slice brings it up on the `Menu Viewer`, allowing you to edit that particular slice. Clicking on a different disk slice on the `Form Viewer` lets you edit it on the `Menu Viewer`.

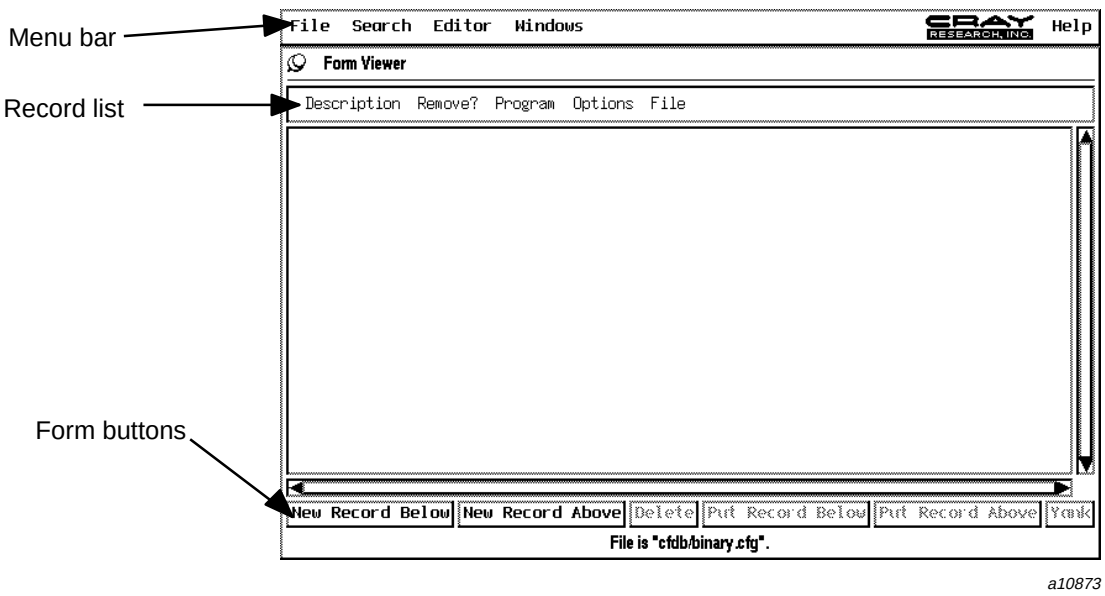


Figure 7. Form menu

The `Form Viewer` window contains a record list, a menu bar, and form buttons. The current record is highlighted in the record list. You can select another record by clicking on the appropriate entry. All of the form buttons operate on the current highlighted record.

The menu bar contains `Form Viewer` pull-down menus. Menu options that are unique to the `Form Viewer` are as follows:

`Save` (on File menu)	Updates the form file if needed, without exiting the form.
`Forward` (on Search menu)	Searches the form list for the specified string. The search starts from the current record in a clockwise direction.
`Backward` (on Search menu)	Searches the form list for the specified string. The search starts from the current record in a counterclockwise direction.
`Repeat` (on Search menu)	Repeats the last search in the same direction as before.
`Vi Emacs Emacs -nw Preference editor` (on Editor menu)	Edits the current form list using the specified editor. Note that the `Emacs -nw` option keeps `emacs`(1) from creating a separate window.
`Form Viewer Commands` (on Help menu)	Displays help file information about form menu bars and form buttons.

The form buttons are as follows:

`New Record Below`	Creates a new record, with default values in the fields, after the current line.
`New Record Above`	Creates a new record, with default values in the fields, before the current line.
`Delete`	Deletes the current line.
`Put Record Below`	Puts a yanked or deleted line after the current line.
`Put Record Above`	Puts a yanked or deleted line before the current line.
`Yank`	Copies the current line into a buffer for later use.

4.15.3.7 Help Menu

The Help pull-down menu contains the following options:

`Help`	Displays online help for the current menu screen.
`About`	Displays copyright and version information.
`Man Page`	Accesses UNICOS/mk online man pages.
`WhereAmI Viewer`	Displays the location of the current menu within the menu system and the path through the menu system that you have traversed. You can go to a previous menu by clicking on the appropriate entry.
`Tail install.log`	Tails the log file `install.log`.
`Viewer Commands` (in menu menu bar)	Displays help file information about menu bars, menu buttons, and input keys.
`Tree Viewer Commands` (in Tree menu bar)	Displays help file information about Tree menu bars, menu buttons, and input keys.
`Form Viewer Commands` (in Form Viewer menu bar)	Displays help file information about Form Viewer menu bars, menu buttons, and input keys.

4.15.4 Menu Buttons

The menu buttons are as follows:

`Main Menu`	Returns to the main (top-most) menu.
`Parent Menu`	Returns to the menu one level up in the menu tree.
`Previous Menu`	Returns to the menu you just came from in the menu tree. This button is disabled if the last menu was a form.

4.15.5 Input Keys

Table 4 describes the input keys used to enter text from the keyboard into the input area (for the location of the input area, see Figure 5, page 58).

Table 4. Input keys

Key	Action
CONTROL-a	Moves cursor to beginning of input buffer.
CONTROL-b Left arrow	Moves cursor backward 1 character (nondestructive).
CONTROL-e	Moves cursor to end of input buffer.
CONTROL-f Right arrow	Moves cursor forward 1 character (nondestructive).
CONTROL-h Del BackSpace	Deletes previous character (to left of cursor).
CONTROL-k	Deletes from current cursor location to end-of-line.
CONTROL-u	Deletes entire input line.
CONTROL-w	Deletes previous word (to left of cursor).
RETURN ENTER CONTROL-j CONTROL-m	Verifies current input and accepts it, if valid.
INSERT	Inserts new text to right of cursor.

4.15.6 Changing Interface Colors

If desired, you can change the foreground (text) and background (panels) colors. The simplest way to change the color is to customize the colors of the menu system program (inmenu).

Add the following lines to your .Xdefaults file:

```
inmenu*foreground      black
inmenu*background      white
```

or

```
inmenu*foreground      rgb:00/00/00
inmenu*background      rgb:ff/ff/ff
```

4.15.7 Printing a Screen Dump

We recommend using your local workstation to print out a dump. Either of the following command lines may be used to print screen dumps.

Generic `xwd`:

```
xwd | xpr -dev ps -gray 4 | lpr
```

Version that works with `tvtwm`:

```
xwdv | xpr -dev ps -gray 4 | lpr
```

After entering the preceding command, click the mouse on the window to complete the screen dump and start the printing.

Note: If your printed screen dumps are hard to read and do not have enough contrast, using the following command will often produce a legible printout:

```
xwd -add 10
```

Boot-Time Configuration: the Auto-Edit Feature [5]

This chapter describes how the UNICOS/mk system uses the *auto-edit* feature to obtain the correct configuration when the system is started.

The UNICOS/mk system uses both a software and a hardware configuration.

- The software configuration describes the mainframe and its peripherals in an ideal state, when all PEs are working, and no hardware has been disabled. The software configuration is in the text file `/opt/os/1.0/config/param.conf`, or in a location specified in the `/opt/config/options` file, or in a location specified by the `bootsys -p` or `-P` options. You can modify this file directly using a text editor.
- The hardware configuration describes the actual state of the mainframe, including disabled PEs. The hardware configuration is in the text file `/opt/diag/config/t3e_config`. This hardware configuration is created and changed with the `t3ems`(8) diagnostic utility.

Note: For information on using the `t3ems`(8) diagnostic utility to disable PEs, see Chapter 7, page 101.

When the UNICOS/mk system is booted, the software and hardware configurations are compared, and the software configuration is edited to reflect any disabled PEs. This is the auto-edit feature.

This chapter includes the following information on the auto-edit feature:

- Necessary configuration file changes
- Dependencies among software components
- The auto-edit algorithm
- Sample console output

In addition, this chapter includes a section describing how to edit the configuration file manually in case it is necessary to boot the UNICOS/mk system without the auto-edit feature.

5.1 Configuration File Changes

The changes to the configuration file that are necessary to run the auto-edit feature were first implemented in the UNICOS/mk 1.5.2 release. If you have already made these changes, you can skip this section.

A one-time change is required for any UNICOS/mk configuration files. The required changes are to replace the `io_contr_lpe` field with the new `io_contr_ppe` field. The optional (recommended) changes are to add the new `min_cmd` and `min_app` parameters to the configuration file, as described below.

5.1.1 Boot Options Flag

In a UNICOS/mk system, auto-edit is enabled by default. You can prevent auto-edit from running by performing a logical OR operation with the value 0x100000, as follows:

```
boot_info {
    boot_options = 0x100000;
}
```

If the only boot option set is 0x100000, you may use the symbol `BOOT_OPT_DO_AUTOEDIT`.

When you disable the auto-edit feature, you must edit the configuration file manually to reflect any disabled PEs. This procedure is described in “Booting without auto-edit”, Section 5.5, page 78.

5.1.2 Configuration File Parameters

There are two parameters that are used to control the behavior of auto-edit. They are:

- the requested minimum number of command PEs (`min_cmd`)
- the requested minimum number of application PEs (`min_app`)

The following examples show how these parameters are specified. In these examples, we use a T3E LC288 (with a total of 312 PEs). This system might be configured normally with 8 OS PEs, 40 command PEs (including redundant PEs), and 272 application PEs.

Example 1:

```
mpp {
   min_cmd = 20;
}
```

This `min_cmd` value requires that at least 20 command PEs are available. This can be important on systems with a lot of interactive users. The number may be set lower, for example, if there will be only a small number of batch jobs; this is at the discretion of the system administrator. There must be at least one command PE.

Example 2:

```
mpp {
   min_app = 256;
}
```

This `min_app` value requests that at least 256 application PEs are available. The request will be honored, so long as there are enough command PEs.

If one or both of these parameters are not specified in the configuration file, or if the number specified is out of range, then the default values will be assumed. The default value for both parameters is all of them.

See “The auto-edit algorithm,” Section 5.3, page 72, for details on how these parameters are used.

5.1.3 Changed I/O Controller Specification

Before the UNICOS/mk 1.5.2 release, each active GigaRing connection (I-chip) in the system was named by specifying the logical PE number (LPE) of one of the four PEs on the same board, for example:

```
routes[0] {
   io_contr_lpe = 0x134;
}
```

In a UNICOS/mk system this field is now filled in for you by auto-edit at boot time, based on a field in the configuration file which must be provided by the system administrator. It is a physical PE number, for example:

```
routes[0] {
   io_contr_ppe = 0x80200;
}
```

This is the physical PE number (PPE) that corresponds to the LPE number above. There are two benefits to this change. First, the LPE number can be derived from the PPE number, even if the LPEs were renumbered as a result of disabled PE(s). Second, if the specified PE is not available, auto-edit will try each of the other three PEs on the same board, any of which will work as I/O controllers. This will fail only if the I-chip is unreachable (all four PEs on the board are unavailable).

5.2 Dependencies

If you are running UNICOS/mk version 1.5.2 or later, you can skip this section.

The UNICOS/mk system, with auto-edit, is intended to run with new configuration files and SWS-ION release 3.01 or later:

- If an old configuration file is used with the UNICOS/mk 1.5.2 release (that is, if the `io_contr_lpe` field is used in the configuration file), then you must continue to edit the `io_contr_lpe` value yourself to reflect any disabled PEs.
- If an old SWS-ION release is used with the UNICOS/mk 1.5.2 release (that is, SWS-ION before 3.0.1), then you must use an old configuration file, with `io_contr_lpe` fields, and without `io_contr_ppe` fields.

You should not attempt to use new configuration files (with the `io_contr_ppe` field) with older versions of the UNICOS/mk system (before 1.5.2).

You may use the SWS-ION release 3.01 or later with older UNICOS/mk system and/or older configuration files, subject to other constraints in the SWS-ION release notes.

If you are upgrading from an earlier release of the operating system, check to see that the following executable files were replaced on the SWS by the install procedure:

```
/opt/CYRIos/bin/csp
/opt/CYRIos/bin/csv
```

5.3 The Auto-edit Algorithm

There are four steps to the auto-edit algorithm:

1. Go through the `t3e_config` file looking for disabled PEs.

2. Go through the configuration file, looking at per-PE information.
3. Go through the configuration file, looking at the GigaRing routes.
4. Go through the configuration file one last time for a completeness check.

These steps, and the actions taken at each step, are described in the following sections.

5.3.1 Step 1: Go through the `t3e_config` File

The first step of the auto-edit algorithm is to go through the `t3e_config` file looking for disabled PEs. There are two ways to disable a PE: by disabling either the node or the PE link (refer to "Downing a PE," Chapter 7, page 101, for details of this procedure). Any disabled PEs are marked in the configuration file with the `PE_NOBOOT` flag. If there are any existing `PE_NOBOOT` flags in the configuration file, they are honored; but a warning message is printed on the console for each.

5.3.2 Step 2: Go through the Configuration File for Per-PE Information

The second step of the auto-edit algorithm is to go through the configuration file, looking at the per-PE information (the `pe_t` structure). Disabled PEs may be swapped with enabled PEs, as needed, to satisfy a set of criteria, described below for each PE type.

Note that as of UNICOS/mk 1.5.2 there are two new fields in the configuration file: `min_cmd` and `min_app`. The system administrator may use them to request a minimum number of command and application PEs, respectively. The default in each case, if either `min_cmd` or `min_app` is not specified, is all of them. The `min_cmd` value takes precedence over the `min_app` value, if it is not possible to satisfy both requests. The values of the `min_cmd` and `min_app` fields can affect the actions that the auto-edit algorithm performs in swapping OS and Command PEs, as described below.

5.3.2.1 OS PEs

A UNICOS/mk system cannot run with disabled OS PEs. For each disabled OS PE, the algorithm uses the `find_pe` function to find an enabled command or application PE that it can swap, as described below. If there are any disabled OS PEs at the end, the system will panic.

5.3.2.2 Command PEs

For each disabled command PE, the algorithm uses the `find_pe` function to find an enabled command or application PE that it can swap. See "The `find_pe` function", Section 5.3.2.4, page 74 for further information.

If the `min_app` value has not been reached (that is, if there is more than the minimum number of enabled application PEs) then the application PE will be sacrificed in order to enable the command PE.

Otherwise, if the `min_app` value has been reached (that is, if there are too few enabled application PEs), then there are two possibilities:

- If the `min_cmd` value has been reached (that is, if there are too few enabled command PEs), then the application PE will be sacrificed in order to enable the command PE. Therefore, the `min_cmd` value takes precedence over the `min_app` value.
- Otherwise, if the `min_cmd` value has not been reached (that is, if there is more than the minimum number of command PEs), then the command PE will remain disabled, in order to preserve as many application PEs as possible.

5.3.2.3 Application PEs

The application PEs that remain enabled at the end of this procedure will have contiguous logical PE numbers, beginning at zero.

5.3.2.4 The `find_pe` Function

The `find_pe` function looks first for a candidate on the same board (on liquid-cooled systems only). It then looks for PEs on other boards with similar characteristics, such as memory size. Finally, the `find_pe` function steps through all the PEs in the system, from highest to lowest logical PE number, looking for a candidate. A candidate for a swap is validated to be sure that it is enabled, and of the correct type. A command or application PE can be sacrificed to replace an OS PE. An application PE can be sacrificed to replace a command PE.

5.3.3 Step 3: Go through the Configuration File for GigaRing Routes

The third step of the auto-edit feature is to go through the configuration file, looking at the GigaRing routes (the `scxroute_t` structure).

If a route is specified with an I/O controller logical PE number (`io_contr_lpe`) then that route is left alone. No attempt will be made to adjust the `io_contr_lpe` value. This is provided for compatibility, and for cases where the auto-edit function is disabled (using the `boot_options` flag).

Any one of the four PEs on the board with the I-chip may be used as the I/O controller. For example, if the supplied `PWHO` is 0x010203, then we will try it and 0x010202, 0x010103, 0x010102, in that order. A disabled PE cannot be used as an I/O controller. In addition, the `IO` (I/O link disable) flag must not be set for an I/O controller. See Section 7.2.1, page 106, for a note about the `IO` flag.

If none of the PEs on a board can be used as an I/O controller, then that ring cannot be used. If that ring is required for system operation, then there are the following options:

- Use the `t3ems`(8) diagnostic utility to make sure that the `IO` flag is not set on the PEs that you are trying to use for I/O. See Chapter 7, page 101 for instructions on using `t3ems`(8).
- Replace the module, or swap it with another module, so that there is at least one working PE on the board. You may need to use the `t3ems`(8) diagnostic utility to change the characteristics of some PEs as a result, but you will not need to change the configuration file. See Chapter 7, page 101 for instructions on using `t3ems`(8).
- Connect the ring to another I-chip, and change the configuration file to reflect a new physical PE number for that ring's I/O controller.

5.3.4 Step 4: Configuration File Completeness Check

The fourth and last step of the auto-edit algorithm is to go through the configuration file one last time, counting PEs, and making a completeness check. If it looks as if there was a problem, one or more warning messages are printed on the console.

5.4 Sample Console Output from Auto-edit

In the following example, two PEMs have been removed from a Cray T3E LC288 (with 312 total PEs). The removed PEMs are slots 6 and 22, both in cabinet 0.

In addition to the sixteen downed nodes, there are an additional twelve PE links that must be disabled, according to the `t3ebrd` script in the `t3ems`(8) diagnostic utility.

So we have a total of 284 PEs available for this boot. The console output in this example has been simplified by removing the time stamps and other identification from each line.

Disable:

```
Begin auto-edit
    t3e_config phys_npes=312
    Setting PE_NOBOOT (node),    LPE 0x130, PPE 0x40002
    Setting PE_NOBOOT (node),    LPE 0x131, PPE 0x40003
    Setting PE_NOBOOT (node),    LPE 0x132, PPE 0x40102
    Setting PE_NOBOOT (node),    LPE 0x133, PPE 0x40103
    Setting PE_NOBOOT (node),    LPE 0x12c, PPE 0x40202
    Setting PE_NOBOOT (node),    LPE 0x12d, PPE 0x40203
    Setting PE_NOBOOT (node),    LPE 0x12e, PPE 0x40302
    Setting PE_NOBOOT (node),    LPE 0x12f, PPE 0x40303
    Setting PE_NOBOOT (PE link), LPE 0x118, PPE 0x40400
    Setting PE_NOBOOT (PE link), LPE 0x119, PPE 0x40401
    Setting PE_NOBOOT (PE link), LPE 0x11a, PPE 0x40500
    Setting PE_NOBOOT (PE link), LPE 0x11b, PPE 0x40501
    Setting PE_NOBOOT (PE link), LPE 0x114, PPE 0x40600
    Setting PE_NOBOOT (PE link), LPE 0x115, PPE 0x40601
    Setting PE_NOBOOT (PE link), LPE 0x116, PPE 0x40700
    Setting PE_NOBOOT (PE link), LPE 0x117, PPE 0x40701
    Setting PE_NOBOOT (PE link), LPE 0x128, PPE 0x50000
    Setting PE_NOBOOT (PE link), LPE 0x12a, PPE 0x50100
    Setting PE_NOBOOT (PE link), LPE 0x124, PPE 0x50200
    Setting PE_NOBOOT (PE link), LPE 0x126, PPE 0x50300
    Setting PE_NOBOOT (node),    LPE 0x120, PPE 0x60000
    Setting PE_NOBOOT (node),    LPE 0x121, PPE 0x60001
    Setting PE_NOBOOT (node),    LPE 0x122, PPE 0x60100
    Setting PE_NOBOOT (node),    LPE 0x123, PPE 0x60101
    Setting PE_NOBOOT (node),    LPE 0x11c, PPE 0x60200
    Setting PE_NOBOOT (node),    LPE 0x11d, PPE 0x60201
    Setting PE_NOBOOT (node),    LPE 0x11e, PPE 0x60300
    Setting PE_NOBOOT (node),    LPE 0x11f, PPE 0x60301
```

```
CS: Total PEs:        4 OS,  28 command,  280 application
CS: Disabled PEs: -   0 OS,  24 command,    4 application
CS:                 ---     ---          ----
CS: Enabled PEs:  =   4 OS,   4 command,  276 application
```

Swap PEs to satisfy requested minimums:

```
Requested minimum number of command PEs = 24
Requested minimum number of application PEs = 256
Swap LPEs 0x133 and 0x113
Swap LPEs 0x132 and 0x112
Swap LPEs 0x131 and 0x111
 Swap LPEs 0x130 and 0x110
Swap LPEs 0x12f and 0x10f
Swap LPEs 0x12e and 0x10e
Swap LPEs 0x12d and 0x10d
Swap LPEs 0x12c and 0x10c
Swap LPEs 0x12a and 0x10b
Swap LPEs 0x128 and 0x10a
Swap LPEs 0x126 and 0x109
Swap LPEs 0x124 and 0x108
Swap LPEs 0x123 and 0x107
Swap LPEs 0x122 and 0x106
Swap LPEs 0x121 and 0x105
Swap LPEs 0x120 and 0x104
Swap LPEs 0x11f and 0x103
Swap LPEs 0x11e and 0x102
Swap LPEs 0x11d and 0x101
Swap LPEs 0x11c and 0x100
Reached minimum number of application PEs:
Command PE 0x11b will remain disabled
Reached minimum number of application PEs:
    Command PE 0x11a will remain disabled
Reached minimum number of application PEs:
Command PE 0x119 will remain disabled
Reached minimum number of application PEs:
Command PE 0x118 will remain disabled
```

Calculate I/O controller LPE numbers, avoiding disabled PEs:

```
routes[0].io_contr_ppe=0x80200
    Using PPE 0x80200 for I/O controller, routes[0]
    Setting routes[0].io_contr_lpe=0x134
    routes[1].io_contr_ppe=0x30204
```

```
    Using PPE 0x30204 for I/O controller, routes[1]
    Setting routes[1].io_contr_lpe=0x74
    routes[2].io_contr_ppe=0x40200
    Using PPE 0x40200 for I/O controller, routes[2]
    Setting routes[2].io_contr_lpe=0x84
    routes[3].io_contr_ppe=0x40204
    Using PPE 0x40204 for I/O controller, routes[3]
    Setting routes[3].io_contr_lpe=0x8c
    routes[4].io_contr_ppe=0x20300
    Using PPE 0x20300 for I/O controller, routes[4]
    Setting routes[4].io_contr_lpe=0x46
    routes[5].io_contr_ppe=0x20304
    Using PPE 0x20304 for I/O controller, routes[5]
    Setting routes[5].io_contr_lpe=0x56
    routes[6].io_contr_ppe=0x50300
    Can't use PPE 0x50300 for I/O (IO link)
    Using PPE 0x50301 for I/O controller, routes[6]
    Setting routes[6].io_contr_lpe=0x127
```

Final summary:

```
CS: Total PEs:        4 OS,  28 command,  280 application
    CS: Disabled PEs: -   0 OS,   4 command,   24 application
    CS:                 ---     ---          ----
    CS: Enabled PEs:  =   4 OS,  24 command,  256 application
    End auto-edit
```

# # #

5.5 Booting without Auto-edit

It is possible to boot the UNICOS/mk system without the auto-edit feature, although this should be a rare occurrence. If this is necessary, you must edit the UNICOS/mk configuration file manually to account for discrepancies between the software and hardware configuration after PEs have been disabled.

Use the following procedure to disable the auto-edit feature and edit configuration file.

1. Set the `boot_options` flag to disable auto-edit. You must perform a logical OR operation on the value 0x100000 with any other boot options

that have been selected. If this is the only boot option, the configuration file line will look like this:

```
boot_info {
    boot_options = 0x100000;
}
```

If the only boot option set is 0x100000, you may use the symbol BOOT_OPT_DO_AUTOEDIT.

2. Edit the PE configuration.

   You must modify the UNICOS/mk configuration file to tell the system to avoid the disabled PE or node. Whether you are disabling the PE or the entire node (PE and router chip), you will need to locate the corresponding LPE entry in the configuration file. Using the newly created t3e_config file, find the LPE number of each PE that is disabled. For example, if physical PE number 0x00200 has been downed, find its new logical PE number in the new t3e_config file.

   If the disabled PE is an OS PE, you will need to swap this PE with a command PE and then disable it, since you cannot boot the system if an OS PE is disabled. For example, if the t3ems(8) program makes the downed LPE 0x10b (as it will on a 272 PE system) and you have PEs 10a, 10b, and 10c as OS PEs in your configuration file, you must make 10b a command PE and use another PE, such as 10d, as an OS PE. The system must still have at least one command PE.

   For example, for LPE number 0x109 (decimal 265) the entry might look like the following:

```
pe[265] {
    flags = 0;
    pe_type = PE_TYPE_CMD;
    actor_list_index = 0;
}
```

   Change the flags field to include the PE_NOBOOT value:

```
pe[265] {
   flags = PE_NOBOOT;
   ...
}
```

Note: The `pe_index` and `hz` fields above do not need to be in the configuration file if you are running the UNICOS/mk 1.6 release (or later) and you are running SWS-ION release 3.4 (or later).

3. Edit the I/O configuration.

   When the auto-edit feature is disabled, you must remove any `io_contr_ppe` fields from the parameter file, and instead use the `io_contr_lpe` field.

   The logical PE numbers may change when the `t3ems`(8) utility renumbers all logical PEs. This can make rings inaccessible because they now contain erroneous (outdated) `io_contr_lpe` values. If the logical PE number of any ring's I/O control PE has changed after renumbering, you must update that ring's corresponding `io_contr_lpe` value in the configuration file with the new logical PE number.

   Similarly, if you downed a node which is specified as an `io_contr_lpe` value, you must update this field in the configuration file to contain a logical PE number for a working PE on the same module (on an air-cooled system) or 4-PE half-module (on a liquid-cooled system).

Cray T3E System Reconfiguration Procedure [6]

This chapter describes how to reconfigure the software components of a Cray T3E system when you upgrade the number of PEs in your system.

To explain the steps required for a Cray T3E system upgrade, the sections in this chapter use as an example a 264 PE Cray T3E machine with 2 GigaRing channels and two MPNs that is to be upgraded to a 272 PE Cray T3E machine with 4 GigaRing channels and four MPNs. This system configuration is shown in the figure below:

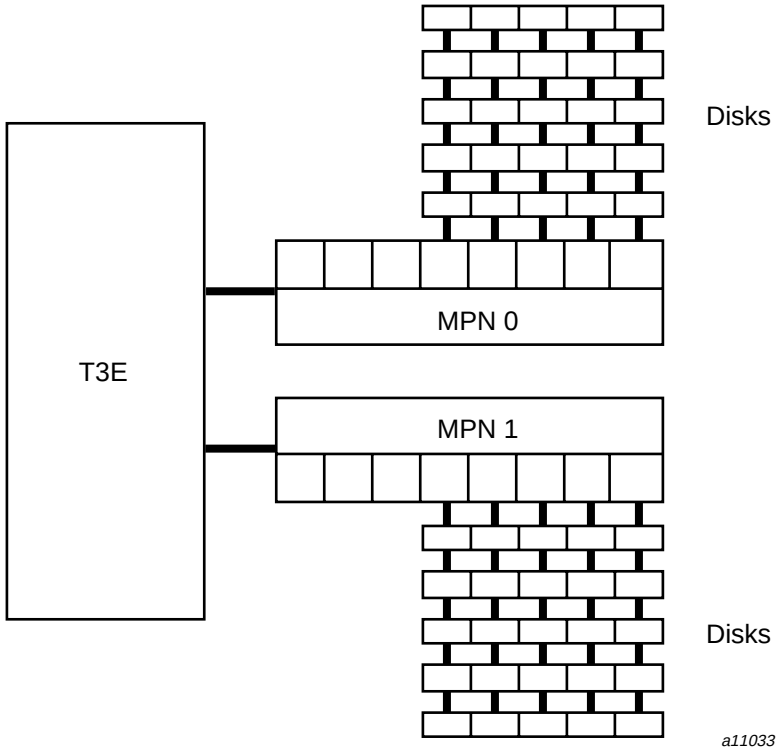


Figure 8. Sample Cray T3E Configuration

There are a total of 40 DD–314 disk drives on this system. The upgrade will add the extra PEs. This will require a new configuration file with an increased PE count and PE definition list. It will also require modifications to the options and topology files located on the SWS. The following diagram illustrates the new system configuration:

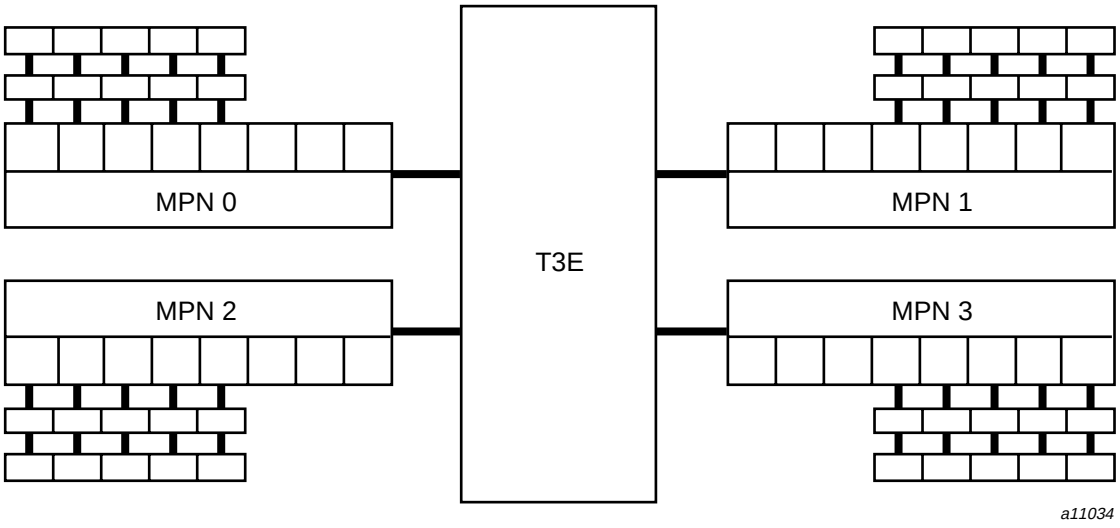


Figure 9. Sample Cray T3E Upgraded Configuration

This chapter covers the following areas you need to consider when reconfiguring a Cray T3E system:

- Hardware components
- Locating the logical PE number for the boot PE
- Software components
- General layout considerations
- Assigning servers
- Adding GigaRing nodes
- Adding additional PEs

6.1 Hardware Components

Before updating your system to account for additional PEs, you must plan out the various elements of your system. The first step is to determine what characterizes the hardware components on your system.

You should consult your hardware engineer to find out the following information about your system, which you can use when making decisions about assigning your PEs:

- How many PEs are in your system
- The memory size of each individual PE
- The processor speed of each individual PE

You can run the `t3ems`(8) diagnostic utility to verify that this information is set correctly in the `t3e_config` file.

Your hardware engineer should be able to provide you with the following information that you will need when you specify your system configuration:

- The logical number (`LWho`) of the boot PE. For information on using `t3ems`(8) to locate the boot PE, see Section 6.2.
- GigaRing routing information, which includes the following:
  - The GigaRing number to which each I/O controller node is connected.
  - The physical PE number that is used to reference this I/O controller node.
  - The logical PE number that is used to reference this I/O controller node.
  - The GigaRing node address of this I/O controller noder.

  In addition, you will need to determine the following software configuration values:

  - The name of the OS PE that contains the packet server used to receive all the interrupts from this I/O controller node.
  - The name of the OS PE which contains the packet server used to send messages out via this I/O controller node. This name is the same as the name of the OS PE that contains the packet server used to receive all the interrupts from this I/O controller node.

    For more information on GigaRing routing, see Section 8.1 and Section 8.2.

– Packet server routing information. For more information on packet server routing, see Section 8.3 and Section 8.4.

Whenever you change a parameter in the `t3ems` `System Configuration` dialog (for example, system type or number of PEs), `t3ems`(8) will regenerate the entire system configuration parameters for the newly defined system. All information about your current system is overwritten with default values. Thus, a default value of 0 is used to fill the `route type`, `DRAM type` and `memory config` fields of your new system.

If the 0 default value is not correct for your system, the first thing to do after setting the system type (or other parameter that causes the entire system to be recomputed) is to display the `PE Configuration` dialog, select all PEs, and set the correct values for those fields on your system. Then save the configuration by selecting `Save` from the `System Configuration` dialog.

Caution: Always check the `route`/`dram`/`memory` configuration parameters when using `t3ems`(8) to regenerate your system. For a description of the `t3e_config` file, see Chapter 11, page 193.

6.2 Locating the Logical PE Number for the Boot PE

If you do not know the logical PE number of the boot PE for your system, you can locate it by using the `t3ems`(8) utility on the SWS workstation.

1. Log into the SWS workstation as `craydiag` and start the `t3ems`(8) utility.

   ```
   sws$ /opt/CYRIdiag/t3e/bin/t3ems
   ```

2. From the main window, select `PE Configuration` by clicking with the right mouse button on the `Config` button menu. The boot PE is the PE that is highlighted in the section at the bottom of the screen where all PEs are listed. For an example of a PE configuration display, see Figure 10, page 85.

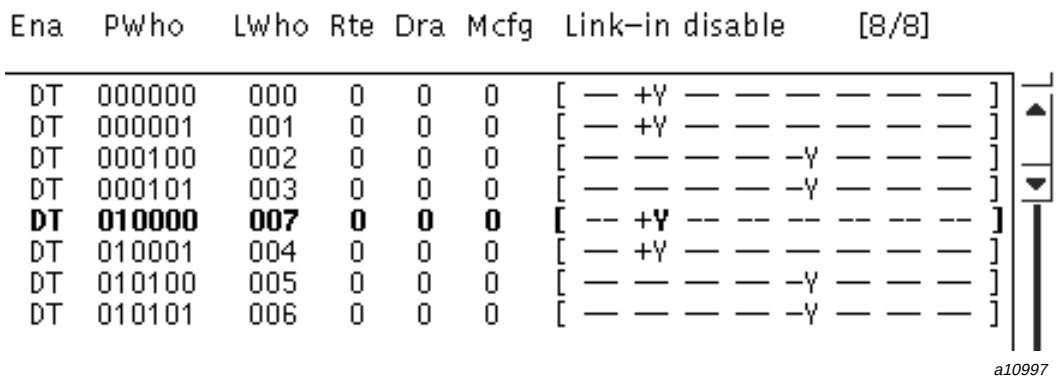


Figure 10. PE Configuration (Boot PE Is Highlighted)

6.3 Software Components

After you have determined the physical attributes of your system, you can plan the layout of your system. You will need to determine the following things:

- How many OS PEs does your system require? You may need to increase this number to account for increases in the I/O or system call load.
- How many command PEs does your system require? In general, you should increase the number of command PEs as the number of interactive system users increases.
- How many application PEs does your system require?
- Which servers will run on which PEs? For information on assigning servers to PEs, see Section 6.5.
- Do additional server routes need to be defined? You must ensure that I/O controller nodes are connected to GigaRing nodes. For information on configuring server routes, see Chapter 8, page 117.

6.4 General Layout Considerations

After considering your system's hardware capabilities and your system requirements, you should map out your system layout. You should note that the logical numbers for all application PEs must be contiguous.

6.5 Assigning Servers

For each OS, command, and application PE in the system there must be a corresponding entry in the `pe[]` structure in the configuration file. Among other things, each entry contains an `actor_list_index` parameter that defines the actors, or servers, that are used for each type of PE.

The following server lists are supplied in the default configuration file:

- A default server list for command PEs
- A default server list for application PEs
- A default OS PE server list for systems with one OS PE
- Default OS PE server lists for systems with two OS PEs

Caution: Do not change the default templates provided in the configuration file.

The following sections provide illustrations of the default server lists.

6.5.1 Command PE Server List

The following shows the default server list for command PEs. As a rule, every command PE needs to run the `kernel`, `PM`, `fsa`, and `em` servers.

```
/* pe_actors index 0 -- Command PEs */
pe_actors[0] {
   actor_name[0] = "kernel";
   actor_name[1] = "PM";
   actor_name[2] = "fsa";
   actor_name[3] = "em";
}
```

6.5.2 Application PE Server List

The following shows the default server list for application PEs. As with command PEs, every application PE needs to run the `kernel`, `PM`, `fsa`, and `em` servers.

```
/* pe_actors index 1 -- Application PEs */
pe_actors[1] {
    actor_name[0] = "kernel";
    actor_name[1] = "PM";
    actor_name[2] = "fsa";
    actor_name[3] = "em";
}
```

6.5.3 OS PE Server List

The following shows the default server list for an OS PE on a system with only one OS PE.

Note: Do not use the following template for any of the OS PEs on a system with two or more OS PEs.

```
/* pe_actors index 2 -- OS PE -- for single-OS-PE system */
pe_actors[2] {
    actor_name[0] = "kernel";
    actor_name[1] = "PM";
    actor_name[2] = "fsa";
    actor_name[3] = "log";
    actor_name[4] = "info";
    actor_name[5] = "em";
    actor_name[6] = "config";
    actor_name[7] = "packet";
    actor_name[8] = "disk";
    actor_name[9] = "tty";
    actor_name[10] = "netdev";
    actor_name[11] = "file";
    actor_name[12] = "socket";
    actor_name[13] = "alm";
    actor_name[14] = "grm";
    actor_name[15] = "GPM";
    actor_name[16] = "ipcm";
    actor_name[17] = "UP_INIT";
    actor_name[18] = "sio";
    actor_name[19] = "tape";
}
```

The following shows the default server lists for the OS PEs on a system with two PEs. With the exception of the required `kernel`, `PM`, and `em` servers, the

list of servers contained in one OS PE on a single-OS PE system is split between two OS PEs.

```
/* pe_actors index 3 -- OS PE -- two-OS-PE system -- first (boot) OS PE */
pe_actors[3] {
    actor_name[0] = "kernel";
    actor_name[1] = "em";
    actor_name[2] = "PM";
    actor_name[3] = "config";
    actor_name[4] = "info";
    actor_name[5] = "log";
    actor_name[6] = "grm";
    actor_name[7] = "alm";
    actor_name[9] = "GPM";
    actor_name[8] = "tty";
    actor_name[8] = "ipcm";
    actor_name[10] = "UP_INIT";
}
/* pe_actors index 4 -- OS PE -- two-OS-PE system -- second OS PE */
pe_actors[4] {
    actor_name[0] = "kernel";
    actor_name[1] = "em";
    actor_name[2] = "PM";
    actor_name[3] = "packet";
    actor_name[4] = "disk";
    actor_name[5] = "netdev";
    actor_name[6] = "file";
    actor_name[7] = "socket";
    actor_name[8] = "sio";
    actor_name[9] = "tape";
}
```

6.5.4 Rules for OS PE Actor List Configuration

It is always preferable to use one of the predefined actor list configurations included in the configuration file. When creating server lists for each OS PE in the system, observe the following rules:

- The `netdev` and `socket` servers must reside in the same OS PE.
- Each OS PE must run the `kernel`, `em` and `PM` servers.
- The following lists of servers can only have one instance in the entire Cray T3E system.

These servers should be configured on the boot PE:

```
config
grm
log
info
ipcm
UP_INIT
```

These servers can be configured on any OS PE:

```
tty
netdev
socket
alm
GPM
sio
tape
```

- The following servers may have more than one instance in the Cray T3E system. These servers must reside only on OS PEs and their location is determined by how the I/O components (I-chips, IONs, GigaRing channels, disks, file systems, partition cache, etc.) are configured in the system:

  ```
  disk
  file
  packet
  ```

- The packet server can be replicated on different OS PEs with the restriction that there can be at most one packet server per GigaRing connection. For example, if there are 5 GigaRing connections, from 1 to 5 packet servers can be configured on different OS PEs.
- Disk servers can be replicated on every OS PE. Only one disk server is required, however, unless one server is not sufficient to process all of the disk requests.

- A packet server can support at most one local and seven remote disk servers.

  Note that if you are using the alternate path feature in your disk configuration(s), and if you are using different packet servers for the primary and alternate paths, you will be using up some of the allowed remote disk server entries to support the alternate path. Similarly, if you are using different disk servers for the primary and alternate paths, you will be using up some of the allowed remote disk server entries to support the alternate path.

- The `fsa` server should not be used for OS PEs. If it is, the OS PE will not make use of it.

In addition to these rules, the following guidelines can be used to obtain better server performance:

- Since the `sio` server makes use of the disk server that handles the swap device, the `sio` and the disk server for the swap device should reside on the same OS PE whenever possible.
- In the UNICOS/mk system, the `sio` server supports multi-partitioned disk devices. For faster `sio` performance, configure multiple disk partitions each containing two or more striped disks.
- The file and disk servers should reside on the same OS PE, if possible.
- The `disk`, `tape`, `tty`, and `netdev` servers make use of the packet server. Better performance is obtained for each of these servers when they are grouped in the same OS PE as the packet server.

Sometimes the `disk`, `tape`, `tty`, or `netdev` server will reside on a different OS PE from the packet server that it needs to communicate with. When this happens, one `server_routes[]` entry needs to be added to the configuration file to describe each remote packet server communication path.

6.6 Adding GigaRing Nodes

When you add GigaRing nodes to your system, you must modify both the `/opt/config/topology` file in the SWS and the configuration file. The following sections describe the procedure for making these modifications.

6.6.1 SWS Configuration

This section describes the changes that are required for the SWS files to recognize the new GigaRing nodes.

To update the SWS configuration when you add new GigaRing nodes, perform the following procedure:

1. Modify the `/etc/hosts` table.

   First add the IP addresses for the new I/O nodes to the`/etc/hosts` table. In our example, we need to add the IP addresses for MPN2 and MPN3 to the `/etc/hosts` file.

2. Modify the `/etc/bootptab` file.

   Repeat the entries for MPN1 for both MPN2 and MPN3, and then modify the hardware address information. The hardware address is displayed on the front of the MPN when it is booting.

   After updating the `/etc/bootptab` file, restart the `bootpd` daemon to reread this file. This is most easily achieved by sending a `SIGHUP` signal to `inetd`. If this fails, then reboot the SWS. Make sure there is an entry in the `/etc/inetd.conf` file for the `bootpd` process. Once you have modified the `/etc/bootptab` file, you should be able to login remotely to each of the MPNs to test that the MPNs are running correctly.

3. Edit the topology file in `/opt/config/topology` to add the two new GigaRing channels. We need to have a ring number assigned to each new GigaRing channel. The ring number must be an integer between 0 and 127, and must be unique.

   For each ring, we also need a node number for the Cray T3E and the respective MPN to connect to. The node number must be an integer between 1 and 63, and must be unique within a given ring.

   The following example adds sn6304–mpn2 and sn6304-mpn3 to the `/opt/config/topology` file:

```
aRING    ring-0          0                      # ring number
T3E      bnch-0         15       BOOTNODE       # node number for T3E
MPN-1    sn6304-mpn0     1       MAINTENANCE    # node number for MPN

RING     ring-1          1       SKIP           # ring number
T3E      bnch-1          2                      # node number for T3E
MPN-1    sn6304-mpn1     1       MAINTENANCE    # node number for MPN
```

```
RING    ring-2          2       SKIP            # ring number
T3E     bnch-2          2                       # node number for T3E
MPN-1   sn6304-mpn2     1       MAINTENANCE     # node number for MPN

RING    ring-3          3       SKIP            # ring number
T3E     bnch-3          2                       # node number for T3E
MPN-1   sn6304-mpn3     1       MAINTENANCE     # node number for MPN
```

In the above example, the fourth GigaRing channel (ring-3) has a unique ring number of 3. This GigaRing connects to a Cray T3E at node number 2 and to MPN sn6304-mpn3 at node number 1.

From a separate window, log in to each MPN and verify that each connects:

```
wkstn$ rlogin sn6304-mpn0
wkstn$ rlogin sn6304-mpn1
wkstn$ rlogin sn6304-mpn2
wkstn$ rlogin sn6304-mpn3
```

6.6.2 Adding GigaRing Nodes to the Configuration File

After you modify the SWS configuration, the SWS contains information about all the GigaRings channels that are connected to the Cray T3E. However, the Cray T3E does not know about the information in the SWS. The Cray T3E can boot but will not be able to access the new rings. For the Cray T3E to communicate with the new rings, you must modify the configuration file on the SWS.

To add GigaRing nodes to the configuration file, perform the following procedure:

1. Make a copy of the existing configuration file to edit:

   ```
   sws$ cd /opt/CYRIos/release_level/config
   sws$ cp param.conf param.conf.4mpn
   ```

2. Modify the configuration file to add the physical routing information for any required packet server. A `routes[]` entry is required for each ring. For our example, the configuration file already has two MPNs defined in the `scxroute_t` structure. To add the new MPNs, copy the existing structure definitions and change the following fields:

   `io_contr_ppe`

   `gigaring_node_addr`

`ringno`

`intr_ps_name`

`send_ps_name`

The `io_contr_ppe` field must contain the logical number for the PE whose I/O controller node is connected to that GigaRing channel.

Before adding the new GigaRing nodes, our sample configuration file looks like the following:

```
/* scx info - MPN0 */
routes[0] {
    io_contr_ppe = 0x10000;
    gigaring_node_addr = 15; /* node number */
    ringno = 0; /* ring number */
    link_type  = link_scx;
    path_type  = primary;
    intr_ps_name = "ospe_b"; /* PE running packetserver */
    send_ps_name = "ospe_b" /* PE running packet server */
}
/* scx info - MPN1 */
    routes[1] {
    io_contr_ppe = 0x30104;
    gigaring_node_addr = 2;
    ringno = 1;
    link_type  = link_scx;
    path_type  = primary;
    intr_ps_name = "ospe_c";
    send_ps_name = "ospe_c"
}
```

We need to add the following two `routes[]` entries to the configuration file:

```
/* scx info - MPN2 */
routes[2] {
    io_contr_ppe = XXX;
    gigaring_node_addr = 2;
    ringno = 2;
    link_type = link_scx;
    path_type = primary;
    intr_ps_name = "ospe_b";
    send_ps_name = "ospe_b"
}
/* scx info - MPN3 */
routes[3] {
    io_contr_ppe = XXX;
    gigaring_node_addr = 2;
    ringno = 3;
    link_type = link_scx;
    path_type = primary;
    intr_ps_name = "ospe_c";
    send_ps_name = "ospe_c"
}
```

To set the `io_contr_ppe` fields for MPN2 and MPN3, you must find out from your hardware engineer which PEs the new MPNs are attached to.

If you decide to add additional OS PEs to your system, then you might want to modify the `intr_ps_name` and `send_ps_name` fields for MPN2 and MPN3 to point to these new OS PEs, such as `ospe_c` or `ospe_d`. For information on OS PEs, see Section 6.5.3, page 87. For information on adding PEs, see Section 6.7, page 95.

6.6.3 Verifying I/O Controller Node Configuration

When you add additional configured I/O controller nodes, you need to verify that the physical PE number that is used to reference this I/O controller node within the mainframe (`io_contr_ppe`) does not have I/O disabled in the `t3e_config` file. You should also ensure that the I/O port of the other 3 PEs on the same printed circuit board (PCB) are enabled, in case it should prove necessary to use them.

Use the following procedure to enable the I/O attribute for each PE that is used as a reference to an I/O controller node:

1. Log in as `craydiag` to run the `t3ems`(8) diagnostic.
2. From the main window, select `PE Configuration` from the list of `Config` menu buttons.
3. From the `PE Configuration` dialog, locate the line that points to the physical PE number that references the I/O controller.

   ```
   DT  000101   10c   0   0   0   [ -- -- -- -- -- -- -- IO ]
   ```

   If the last field of the `Link-In disable` attributes is set to `IO`, it means that the I/O controller node is disabled for this PE.

   Click this field with the right mouse button to enable the I/O attribute for this PE. The entry for this PE should now look like the following:

   ```
   DT  000101   10c   0   0   0   [ -- -- -- -- -- -- -- -- ]
   ```

4. From the main window, select `System Configuration` from the list of `Config` menu buttons.

   From the `System Configuration` dialog, click on the `Save` button to store this change.

5. Exit the `t3ems`(8) diagnostic.

6.7 Adding PEs

This section describes the procedure to follow to add PEs to a Cray T3E system. When you add PEs, you must modify the configuration file.

In our example, we have a system with 3 OS PEs, 13 command PEs, and 248 application PEs. We want to upgrade this system by adding 8 PEs at an LC cabinet and assigning all 8 new PEs as application PEs.

Table 5 shows the Cray T3E PE configuration before and after the upgrade.

Table 5. PE Configuration

PE Type	Before Upgrade	After Upgrade
Application	000 - 247	000 - 255
Command	248 - 259	256 - 267
OS	260 - 262	268 - 270
Command	263	271

In our example, the boot PE before upgrade is logical PE number 0x104 (decimal 260). After the upgrade, the boot PE will be logical PE number 0x10c (decimal 268).

To upgrade the system, perform the following procedure:

1. If necessary, update the boot PE information contained in the `/opt/config/options` file. The boot PE may be defined through the `-B` option for the boot, dump, and halt actions, as in the following examples:

   ```
   sn6304 boot -B 0x10c -p /opt/CYRIos/1.0/config/sn6304.conf
   sn6304 dump -B 0x10c -p /opt/CYRIos/1.0/config/sn6304.conf -M pelist
   sn6304 halt -B 0x10c
   ```

   In the examples above, the `-B` option defines 0x10c as the logical PE number on which to boot, dump, and halt the system. You can give this value in decimal, octal, or hexadecimal representation. You must update this value to reflect the current boot PE if it has changed.

2. Update the UNICOS/mk configuration file. By default, the UNICOS/mk configuration file resides in the directory `/opt/CYRIos/release_level/config`. Otherwise, you can find its location in the `/opt/config/options` file. You specify the location of the configuration file with the `-p` option of the `boot` and dump commands, as in the following examples:

   ```
   sn6304 boot -B 67 -p /opt/CYRIos/1.0/config/sn6304.conf
   sn6304 dump -B 67 -p /opt/CYRIos/1.0/config/sn6304.conf -M pelist
   ```

   In the example above, the configuration file resides in `/opt/CYRIos/1.0/config/sn6304.conf`.

   Make the following changes to the configuration file:

a. Update the `mf[0].boot_info.num_pes` parameter to specify the new total number of PEs in the system.

```
boot_info {
    num_pes = 272;
}
```

b. If the logical number for the boot PE changed, the `mf[0].boot_info.boot_pe` parameter needs to be updated.

```
boot_info {
    boot_pe = 0x10c;
}
```

The first `routes[]` table entry (which describes the I/O controller node that is local to the boot PE) also needs to be changed to point to the physical PE number for the new boot PE. Find the physical PE number of teh boot PE in the `t3e_config` file. For this example, the new boot PE is 0x30000 .

```
routes[0] {
    io_contr_ppe = 0x30000;
}
```

If there are any additional `routes[]` entries on the old machine, verify that the GigaRing channels are still connected to the same physical PE number.

c. If an I/O controller node is being deleted, then you must remove its `routes[]` entry. You must also remove its corresponding entry in the `mf[0]` structure.

d. Add/modify the `mf[0].mpp.pe[]` entries to reflect the added PEs. The following parameters must be set for each PE that is defined:

```
pe[*] {
   flags = 0;  /* 0 or PE_NOBOOT */
   pe_type = XXX  /* command, app, or os PE */
   actor_list_index = X  /* actor list number */
   pe_name = "ospe_x"; /* OS PEs only */
}
```

The * character signifies the logical PE number or range of logical PE numbers.

Starting with the UNICOS/mk 2.0 release, the pe_index and hz fields in the pe_t structure are no longer used by the Cray T3E mainframe. They have been redundant for some time and are now ignored.

However, these two fields must continue to be specified in the configuration file as part of the pe_t structure definition, due to an SWS-ION dependency. These two fields may be removed from the body of the configuration file after you have installed SWS-ION release 3.6 or later.

In addition, each new OS PE will need to define a pe_name field to contain a unique PE name.

Application PEs must be a contiguous chunk of LPEs, so they must have contiguous entries in the pe[] array. OS and command PEs can reside anywhere in this structure.

By convention, the boot PE is at or near the highest LPE number in the system. So we start numbering application PEs at LPE 0, and put the OS PEs near the high end; in between are the command PEs.

For our example, we add the additional 8 PEs at the end of the last application pe[] entry. This means that all entries after this one will need to be increased by 8. Our new pe[] entries will have the following layout:

PE Type	Qty.	pe[] entry
Application	256	pe[0] - pe[255]
Command	12	pe[256] - pe[267]
OS	3	pe[268] - pe[270]
Command	1	pe[271]

This layout can be specified in the configuration file as follows:

```
pe[0:255] {
    flags = 0;
    pe_type = PE_TYPE_APP;
    actor_list_index = 0;
}
```

```
pe[256:271] {
    flags = 0;
    pe_type = PE_TYPE_CMD;
    actor_list_index = 1;
}

pe[268:270] {
    flags = 0;
    pe_type = PE_TYPE_OS;
    actor_list_index = 2;
}

pe[268] { pe_name = "ospe_a"; }

pe[269] { pe_name = "ospe_c"; }

pe[270] {
    pe_name = "ospe_c";
    actor_list_index = 3;
}
```

e. Update the `server_route` entries, if necessary.

By default, if a `server_route` entry is not specified, the device driver servers `disk`, `tty`, `netdev`, and `tape` will use the local packet server to route the request. Therefore, if the local packet server can service all the rings that a server in the same PE require, no `server_routes[]` entries are necessary.

Server routes are needed only if you want to or have to use a packet server on another PE to route the request. For example, if the `tty` server on some OS PE, for example `ospe_a`, does not have a local packet server, you will need to define a `server_routes[]` entry that tells the `tty` server which packet server to use.

```
server_routes[0] {
    ringno = 0;
    server_name = "tty";
    server_pe_name = "ospe_a";
    ps_pe_name = "ospe_c";
}
```

In the above example, the `tty` server in `ospe_a` will use the packet server on `ospe_c`.

3. Run your configuration file through the csp(8) and csv(8) commands located in /opt/CYRIos/bin . These commands check that the syntax and the semantics of the configuration file are correct.

```
SWS$ cd /opt/CYRIos/1.0/config

SWS$ /opt/CYRIos/bin/csp -i param.conf
Parsed file "param.conf" successfully

SWS$ /opt/CYRIos/bin/csv -i param.conf
Parsed file "param.conf" successfully
```

If there are any problems while parsing these files, an error message will be printed to the terminal. Additional information about this problem will also be printed to the /tmp/csp.trace file, as in the following example:

```
SWS$ /opt/CYRIos/bin/csp -i param.conf
csp: unable to parse file param.conf see log file :/tmp/csp.trace

SWS$ tail /tmp/csp.trace
11/08/96 11:35:08 **** Configuration Server Proxy Started As Pid 1629 ****
11/08/96 11:35:15  Parsed file "param.conf" successfully
11/08/96 11:35:15 csp: normal server start of pid 1629.

11/08/96 11:35:15 csp: Configuration Server stopped.
11/08/96 11:36:59 **** Configuration Server Proxy Started As Pid 1633 ****
11/08/96 11:37:02 csp: parse error at line 2873
11/08/96 11:37:02 csp: server start of pid 1633 failed!

11/08/96 11:37:02 csp: Configuration Server stopped.
```

Downing a PE [7]

In general, there are three possible types of PE or route failure on a Cray T3E system:

- Bad PE: The EV5 processor or memory fails, but the router chip is still functioning (messages are still passed through that node). To fix this problem, you must eliminate the PE from the configuration (this is referred to as downing the PE link).
- Bad route: The router chip fails and is unable to route messages. To fix this problem, you must eliminate the PE and its routes from the configuration (this is referred to as downing the node).
- Bad link: A connection between PEs fails, but other connections (routes) still function and the PE can still be used. To fix this problem, you must eliminate the bad link and update the `t3ems`(8) configuration file, but you do not need to map out the PE or route. (This is referred to as downing the torus link).

Caution: Up to and including the UNICOS/mk 1.5.2 release, you cannot disable a torus link unless you also disable the PEs at each end of the link. See FN #2358 for information on this procedure.

To boot with a bad PE or route, use the procedure described in Section 7.1 to remove the processor or node from the configuration.

Note: When a PE has failed because of a persistent hardware failure and a reboot is impractical or inconvenient, you can use the `renumber`(8) command to dynamically renumber one or more Cray T3E logical processing elements (LPEs) to work around a failed PE. For additional information about using the `renumber`(8) command, see the `renumber`(8) man page.

7.1 Mapping out a Bad PE or Router Chip

1. Shut down the UNICOS/mk system. If one or more PEs have failed on a Cray T3E system, the recovery procedure requires a reboot of the UNICOS/mk system. If the system is still running, shut down the UNICOS/mk system with the `/etc/shutdown` script, then halt the system by running the `haltsys` command from the SWS, as user `crayops`.

   ```
   sws$ haltsys
   ```

2. Log into the SWS workstation as `craydiag` and make copies of the original `t3e_config` file so that it will be easy to restore after the hardware has been repaired.

   ```
   sws-craydiag$ cd /opt/CYRIdiag/config
   sws-craydiag$ cp t3e_config t3e_config.date
   ```

3. Run the `t3ems`(8) diagnostic program to map out the bad PE(s). The final result of this step will be a new `t3e_config` file, written by `t3ems`(8). We recommended that you do not edit the `t3e_config` file by hand.

   For further information on using `t3ems`(8) to disable PEs, see Section 7.2, page 105.

   Execute the following command to run `t3ems`(8):

   ```
   sws-diag$ t3ems
   ```

   Use the following procedure to map out the bad PE(s):

   a. From the main window select `System Configuration` by clicking with the left mouse button on the `Config` button menu. In the `System Configuration` dialog, click the `Reload` button with the left mouse button. This rereads the configuration information and applies this information to the current session.

   b. From the main window, click with the right button on the `Config` button menu and select `Pe Configuration` to bring up a dialog showing the configuration for each PE in the system. At the bottom of this dialog is the list of PEs for this system. In this list, locate the PE that is causing the problem.

   c. If you have a bad PE and want to disable the PE while leaving the router chip functional, select (highlight) the PE entry from the list and click on the PE button above the display to disable it. The PE is disabled when `PE` appears in the next to last field of the PE attribute list and the first column displays `D-` (that is, the node is still functional but the PE is disabled). An example is shown in the following figure.

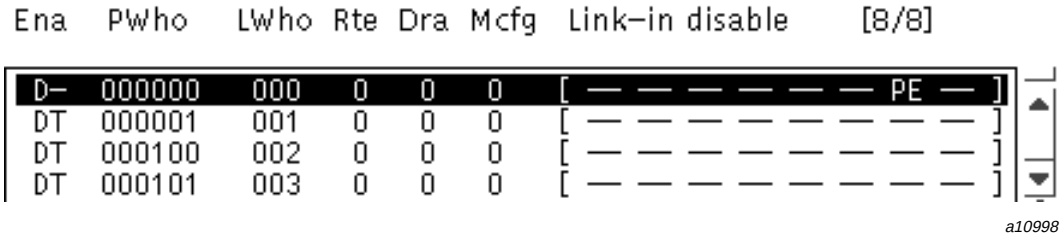


Figure 11. Disabling a PE

d. If you have a bad router chip, disable the entire node by selecting the PE entry from the list and clicking on the <D> button to disable the node. The node is disabled when the D field in the first column has the entry - as shown in the following figure.

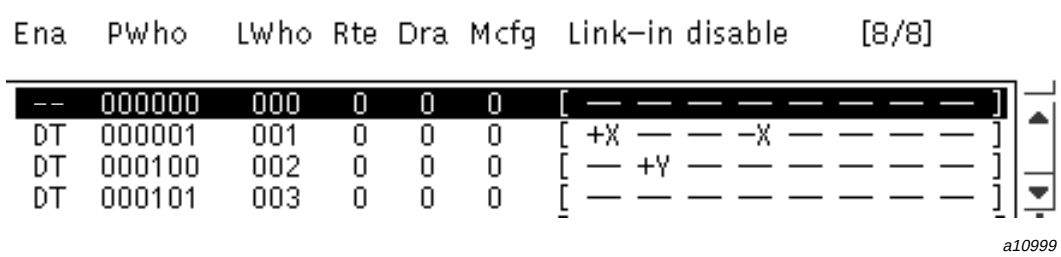


Figure 12. Disabling a Node

e. Click on the Renumber PEs button on the PE Configuration window. This will renumber the logical PE numbers (LWho) onto the physical PE numbers (PWho). Clicking on the Renumber PEs button allows you to see the logical PE renumbering before saving the t3e_config file.

Renumbering the logical PEs preserves the application partition (if there are enough redundant or command PEs to substitute for failed application PEs). If a large number of PEs has failed, you will have to sacrifice some application PEs so that you have enough command PEs to run your workload.

f. In the System Configuration window click the Save button. This will bring up a dialog asking you to confirm this action. Your changes will now be saved in the /opt/CYRIdiag/config/t3e_config file.

g. If you have disabled any nodes, you must check the routes. (It is possible to create configurations that cannot be booted; checking the routes now may save time later.)

Note: This test should not be performed while the UNICOS/mk operating system is up and running. To test the routes fully, the Cray T3E mainframe must be deadstarted by t3ems(8).

To check the routes, run the t3ebrd script in t3ems(8):

- In the main t3ems(8) window, click the left mouse button on the Scripts button. The Load Program window will pop up.
- Scroll down the list of scripts in the Load Program window until you see t3ebrd.pgm. (If you don't see it, click on the System Directory button with the left mouse button.)
- Select (highlight) the t3ebrd.pgm with the left mouse button.
- Click with the left mouse button on the Accept button. The t3ems Program Choice window and the t3ems Script Output window will pop up.
- Select (highlight) the Deadstart Test with the left mouse button.
- Click with the left mouse button on the Done button.
- If there is trouble (see examples below), and the script is able to fix the problem, then you may see a pop-up window that asks "Save Updated Configuration?".
  - Click with the left mouse button on Yes.
  - Click with the left mouse button on the Done button.

h. The following example shows sample output from the t3ems(8) t3ebrd script that uncovered no problems:

```
Testing special routes from boot PE to non-boot PEs...
The special routes are all available.
Testing normal routes between boot PE and non-boot PEs...
The normal routes are all available.
[14:31:45] deadstarting system... (ok)
```

i. The following example shows sample output from the t3ems(8) t3ebrd script after PEM 0.22 was removed from an LC288 system. Eight nodes were disabled as described above. An additional four PE links also had to be disabled by t3ebrd:

```
Testing special routes from boot PE to non-boot PEs...
The special routes are all available.
Testing normal routes between boot PE and non-boot PEs...
The normal routes are all available.
[14:37:50] deadstarting system... (failed)
Testing normal routes for the 4 PE(s) which failed to deadstart...
Some normal routes are unavailable.
Disabling the following 4 PE(s) to resolve route problems:
Disabled PE port for P_WHO 0x50200 (L_WHO 0x12c).
Disabled PE port for P_WHO 0x50300 (L_WHO 0x12e).
Disabled PE port for P_WHO 0x50000 (L_WHO 0x130).
Disabled PE port for P_WHO 0x50100 (L_WHO 0x132).
[14:38:05] deadstarting system... (ok)
```

j. Resolving problems

If there are problems with either the normal or special routes, you will need to take action. For example, you may be able to move the PEM(s) with failed PE(s) elsewhere in the mainframe, so that the working PEs will be able to communicate with each other.

4. Reboot the Cray T3E system, as user crayops:

```
sws-crayops$ bootsys
```

When you boot the system, the auto-edit feature compares the changes you have made to the hardware configuration with t3ems(8) to the software configuration and updates the software configuration. For information on the auto-edit feature see Chapter 5, page 69.

7.2 Additional Considerations

This section provides some additional considerations to keep in mind when using t3ems(8) to disable PEs.

7.2.1 The IO Flag

When the IO flag is set, you disable I/O from a particular PE to the I-chip on the same board. If an I-chip is cabled to a GigaRing route, then typically all four PEs on that board have the IO flag cleared. This is recommended for the UNICOS/mk 1.5.2 release and later, since the system will look at all four PEs on a board to see if any one of them can be used as the I/O controller PE.

An I-chip is inaccessible if all four PEs on its board are disabled (PE link or node) or have the IO flag set. If this I-chip is cabled to a GigaRing route that is not required for system operation, you will still be able to boot the UNICOS/mk system, but you will not be able to access that GigaRing route. If that GigaRing route is required for system operation, you have several options, as described in Chapter 5, page 69.

7.2.2 Missing PEMs

If one or more PEMs are physically removed from the Cray T3E mainframe, then you must disable all nodes on the missing PEMs. You may also need to disable some PE links, based on the results of the t3ebrd script.

However, it is possible that t3ebrd will tell you that it is not possible to boot the resulting configuration. In this case, you may be able to move PEMs around to find a configuration that will boot. For example, in a two-cabinet system, you could use PEMs from cabinet 1 to fill in a large hole in the middle of cabinet 0.

7.3 Changes to the t3ems(8) Diagnostic Utility in SWS-ION Release 3.6

A new version of the t3ems(8) offline diagnostic maintenance system in the SWS-ION 3.6 release changes the way disabled processing elements (PEs) are renumbered (that is, remapped) after they have been disabled. This addresses issues for Cray T3E systems with mixed configurations, such as systems that have PEs with different memory sizes. These changes do not affect systems that do not have mixed configurations.

This section documents how these change to the t3ems(8) offline diagnostic interact with the UNICOS/mk operating system. You do not have to upgrade or change anything in the UNICOS/mk operating system to take advantage of this new version of the t3ems(8) diagnostic system.

7.3.1 Overview of Changes to t3ems(8)

With the SWS-ION 3.6 release, you can specify how many PEs are renumbered in the new Renumbered PEs field. Previously, the default value for renumbering was hard coded. See Section 7.3.2, page 107, for details about how to choose the best value.

The goal is to automate, as much as possible, the management of disabled PEs. You should be able to disable one or more PEs by using the t3ems(8) utility and then simply reboot the UNICOS/mk operating system.

Beginning with the UNICOS/mk 1.5.2 release, the auto-edit algorithm in the operating system handles disabled PEs, so you do not have to edit the UNICOS/mk configuration file manually. Beginning with the SWS-ION 3.6 release, auto-edit correctly handles mixed OS and command PEs.

However, if your system has mixed application PEs and you disable a PE, you may have to change the way t3ems(8) renumbers PEs. If you want to ensure that the partitions of PEs start at specific logical PE numbers, see the examples in Section 7.3.4, page 109, and Section 7.3.5, page 111. If you do not renumber PEs so that partitions start at a specific logical PE number, the system will still boot and run.

With the SWS-ION 3.6 version of the t3ems(8) utility, you disable PEs the same way as with previous versions: select one or more PEs in the PE Configuration window, then click on either the PE button to disable the PE link or the DS button to disable the entire node. After you save the new t3e_config file with a new mapping, the Renumber button will be grayed out (that is, displayed in gray), which means that it is inactive. This inactive state is saved in the t3e_config file so you do not inadvertently renumber PEs. For more information, see Section 7.3.3, page 108.

7.3.2 The Renumbered PEs Field

The t3ems(8) PE Configuration window includes a new numeric field, Renumbered PEs. You can edit this field. In general, you want to renumber application PEs, but you do not want to renumber command or OS PEs, especially if you have mixed-size command or OS PEs. The auto-edit algorithm will try to select an appropriate substitute for a disabled OS or command PE, but if an OS or command PE is renumbered, the auto-edit algorithm cannot select an appropriate substitute.

The correct Renumbered PEs value depends on how you have configured your system. For example, assume that you have a Cray T3E LC416 system that

has a total of 448 PEs, and that you have configured 384 PEs as application PEs. Because a Cray T3E LC416 system has four redundant PEs, four of the command PEs are actually redundant PEs. So a good value for `Renumbered PEs` would be 388, which is the sum of the application and redundant PEs (384+4). For more information, see the examples in Section 7.3.4, page 109, and Section 7.3.5, page 111.

Below the `Renumbered PEs` field is another new numeric field, `Static PEs`, which tells you how many PEs will not be renumbered. You cannot edit this field; it is for information only.

7.3.3 The `Renumber` Button

The `Renumber` button of the `t3ems`(8) `PE Configuration` window has two new menu items, as follows:

- `PEs enabled for renumbering`

  Use this menu item to renumber the PEs after you have disabled one or more PEs. PEs also are renumbered automatically when you click on the `Save` button in the `System Configuration` window.

- `All PEs to original default numbering`

  Use this menu item to go back to the original default mapping before you enable or disable additional PEs.

After you save a new configuration, the `Renumber` button will grayed out (that is, displayed in gray), which means that it is inactive. This inactive state is saved in the `t3e_config` file because when you renumber, you should start from the initial default mapping (that is, you cannot renumber until you return to the initial default mapping).

To disable additional PEs (or to make any change that will require renumbering the PEs), perform the following procedure:

1. Re-enable renumbering by clearing the `Disable PE` renumbering checkbox in the `System Configuration` window
2. Restore the original mapping (numbering) by selecting the `All PEs to original default numbering` menu item under the `Renumber` button in the `PE Configuration` window
3. Enable and/or disable PEs by using the standard procedures.

4. Renumber PEs by selecting the `PEs enabled for renumbering` menu item under the `Renumber` button in the `PE Configuration` window.

7.3.4 One Disabled Application PE Example

This section describes how to use the `Exchange PE/Board Lwhos` feature in `t3ems`(8) to handle a disabled application PE on a Cray T3E system that has mixed-memory-size application PEs. This example applies to any liquid-cooled (LC) system that has more than 256 processors (Cray T3E LC248 and larger systems). The `Exchange PE/Board Lwhos` feature is used once during this procedure.

The hypothetical mainframe in this example is a Cray T3E LC416 system that has a total of 448 PEs, as follows:

- Logical PEs (LPEs) 0x000 through 0x0ff, 256 small application PEs
- LPEs 0x100 through 0x17f, 128 large application PEs
- LPEs 0x180 through 0x1bf, 64 small PEs (OS, redundant, and command PEs)

In this example, the first 388 PEs, from LPE 0x000 through 0x183, have been renumbered (as shown in Section 7.3.2, page 107). 388 is the sum of 256+128+4, which includes all the configured application PEs and the redundant PEs.

The following is a diagram of the default configuration for this example:

```
 0x000                           0x100           0x180
+-------------------------------+---------------+-------+
| 256 small PEs                 | 128 large PEs | 64 PEs|
+-------------------------------+---------------+-------+
```

The remainder of this section describes how to disable and renumber one application PE. Detailed instructions on how to use the `t3ems`(8) utility to perform this procedure are not included; for more information, see the `t3ems`(8) man page.

1. Select and disable the failing PE. The asterisk (*) in the following diagram marks the board where the disabled PE is located (a board contains 4 PEs):

```
 0x000                           0x100           0x180
+------------------+-+---------+---------------+-------+
| 256 small PEs    |*|         | 128 large PEs | 64 PEs|
+------------------+-+---------+---------------+-------+
```

2. Renumber the PEs. Because PEs are remapped in groups of four, all four PEs on the marked board have been moved. In the following diagram, note that the boundary between the small application PEs and the large application PEs has moved. Also, note that the 64 small PEs at the high end have been split into two chunks: one chunk is 60 PEs long (labeled A), and the other chunk is 4 PEs long (labeled B).

```
 0x000                            0x0fc           0x180
+-----------------------------+--------------+-+-+-----+
| 252 small PEs               | 128 large PEs |B|*|  A  |
+-----------------------------+--------------+-+-+-----+
                                                  ^
                                                  |
                Highest renumbered PE is LPE 0x183-+
```

3. Select all four PEs on the first large-memory board and all four PEs on the first small-memory board (that does not have any disabled PEs) above the large-memory PEs. (Before you select these boards you must deselect all PEs.) In the following diagram, the large-memory board is labeled C (LPEs 0x0fc through 0x0ff) and the small-memory board is labeled B (LPEs 0x17c through 0x17f). The board with the disabled PE is still labeled * (LPEs 0x180 through 0x183).

```
 0x000                            0x0fc            0x180
+-----------------------------+-+------------+-+-+-----+
| 252 small PEs               |C|124 large PEs|B|*|  A  |
+-----------------------------+-+------------+-+-+-----+
```

4. Exchange the selected boards. After B and C are exchanged, the configuration is as follows:

```
 0x000                            0x0fc            0x180
+-----------------------------+-+------------+-+-+-----+
| 252 small PEs               |B|124 large PEs|C|*|  A  |
+-----------------------------+-+------------+-+-+-----+
```

   After the exchange, the configuration is 256 small application PEs followed by 128 large application PEs, as shown in the following diagram. At the end of the range of application PEs, 64 PEs still are used for OS, redundant, and command PEs. One of these 64 PEs (a redundant one) is disabled.

```
 0x000                              0x100           0x180
+-------------------------------+--------------+-------+
| 256 small PEs                  | 128 large PEs |*64 PEs|
+-------------------------------+--------------+-------+
```

In summary, the following is a diagram that shows the results of each step in the procedure used in this example:

```
 0x000                            0x100           0x180
+--------------------------------+---------------+-------+
| 256 small PEs                  | 128 large PEs | 64 PEs|
+------------------+-+---------+---------------+-------+
| 256 small PEs     |*|         | 128 large PEs | 64 PEs|
+------------------+-+-------+---------------+-+-+-----+
| 252 small PEs               | 128 large PEs |B|*|  A  |
+-----------------------------+-+-------------+-+-+-----+
| 252 small PEs               |C|124 large PEs|B|*|  A  |
+-----------------------------+-+-------------+-+-+-----+
| 252 small PEs               |B|124 large PEs|C|*|  A  |
+-----------------------------+-+-------------+-+-+-----+
| 256 small PEs                  | 128 large PEs |*64 PEs|
+--------------------------------+---------------+-------+
```

7.3.5 Two Disabled Application PEs Example

This section describes how to use the `Exchange PE/Board Lwhos` feature in `t3ems`(8) to handle two disabled application PEs on a Cray T3E system that has mixed-memory-size application PEs. This example applies to any liquid-cooled system that has more than 256 processors (Cray T3E LC248 and larger systems). The `Exchange PE/Board Lwhos` feature is used three times in this procedure (that is, three pairs of boards will be exchanged).

The hypothetical mainframe in this example is a Cray T3E LC416 system that has a total of 448 PEs, as follows:

- LPE 0x000 through 0x0ff, 256 small application PEs
- LPE 0x100 through 0x17f, 128 large application PEs
- LPE 0x180 through 0x1bf, 64 small PEs (OS, redundant, and command PEs)

In this example, the first 388 PEs, from LPE 0x000 through 0x183, have been renumbered (as shown in Section 7.3.2, page 107). 388 is the sum of 256+128+4, which includes all the configured application PEs and the redundant PEs.

The following is a diagram of the default configuration:

```
 0x000                           0x100           0x180
+-------------------------------+---------------+-------+
| 256 small PEs                 | 128 large PEs | 64 PEs|
+-------------------------------+---------------+-------+
```

The remainder of this section describes how to disable and renumber two application PEs. Detailed instructions on how to use the `t3ems`(8) utility to perform this procedure are not included; for more information, see the `t3ems`(8) man page.

1. Select and disable the failing PEs. In the following diagram, the boards where the disabled PEs are located are marked with * and + (a board contains 4 PEs):

```
 0x000                             0x100           0x180
+-------------------+-+---+-+---+---------------+-------+
| 256 small PEs     |*|   |+|   | 128 large PEs | 64 PEs|
+-------------------+-+---+-+---+---------------+-------+
```

2. Renumber the PEs. Because PEs are remapped in groups of four, all four PEs on each of the marked boards have been moved. In the following diagram, note that the boundary between the small application PEs and the large application PEs has moved. Also, note that the 64 small PEs at the high end have been split into two chunks: one chunk is 60 PEs long (labeled A), and the other chunk is 4 PEs long (labeled B).

```
 0x000                        0x0f8              0x180
+---------------------------+--------------+-+-+-+-----+
| 248 small PEs             | 128 large PEs |B|+|*|  A  |
+---------------------------+--------------+-+-+-+-----+
                                                 ^
                                                 |
                 highest renumbered PE is LPE 0x183-+
```

3. Select all four PEs on the first large-memory board and all four PEs on the first small-memory board (that does not have any disabled PEs) above the large-memory PEs. (Before you select these boards you must deselect all PEs.) In the following diagram, the large-memory board is labeled C (LPEs 0x0f8 through 0x0fb) and the small-memory board is labeled B (LPEs 0x178 through 0x17b). The boards with the disabled PEs are still labeled * and + (LPEs 0x17c through 0x183).

```
 0x000                        0x0f8              0x180
+---------------------------+-+-------------+-+-+-+-----+
| 248 small PEs             |C|124 large PEs|B|+|*|  A  |
+---------------------------+-+-------------+-+-+-+-----+
```

4. Exchange the selected boards. After B and C are exchanged, the configuration is as follows:

```
 0x000                        0x0f8              0x180
+---------------------------+-+-------------+-+-+-+-----+
| 248 small PEs             |B|124 large PEs|C|+|*|  A  |
+---------------------------+-+-------------+-+-+-+-----+
```

5. Pause to examine the current configuration, as follows:

```
 0x000                          0x0fc             0x180
+----------------------------+--------------+-+-+-----+
| 252 small PEs               | 128 large PEs |+|*|  A  |
+----------------------------+--------------+-+-+-----+
```

6. Select all four PEs on the first large-memory board and all four PEs on the first small-memory board (that does not have any disabled PEs) above the large-memory PEs. (Before you select these boards you must deselect all PEs.) In the following diagram, the large-memory board is labeled D (LPEs 0x0fc through 0x0ff) and the small-memory board is labeled E (LPEs 0x184 through 0x187). The boards with the disabled PEs are still labeled * and + (LPEs 0x17c through 0x183).

```
 0x000                          0x0fc             0x180
+----------------------------+-+-------------+-+-+-+---+
| 252 small PEs               |D|124 large PEs|+|*|E| A |
+----------------------------+-+-------------+-+-+-+---+
```

7. Exchange the selected boards. After D and E are exchanged, the configuration is as follows:

```
 0x000                          0x0fc             0x180
+-----------------------------+-+-------------+-+-+-+---+
| 252 small PEs               |E|124 large PEs|+|*|D| A |
+-----------------------------+-+-------------+-+-+-+---+
```

8. Pause to examine the current configuration, as follows. There are 256 contiguous small application PEs, and the large application PEs still must be exchanged.

```
 0x000                           0x100           0x180
+-------------------------------+-------------+-+-+-+---+
| 256 small PEs                 |124 large PEs|+|*|D| A |
+-------------------------------+-------------+-+-+-+---+
```

9. Reselect all four PEs on the large-memory board labeled D in the following diagram (LPEs are now 0x184 through 0x187) and select all four PEs on the first of the two boards that have disabled PEs (labeled + in the following diagram, LPEs 0x17c through 0x17f). The boards with the disabled PEs are still labeled * and + (LPEs 0x17c through 0x183).

```
 0x000                           0x100           0x180
+-------------------------------+-------------+-+-+-+---+
| 256 small PEs                 |124 large PEs|+|*|D| A |
+-------------------------------+-------------+-+-+-+---+
```

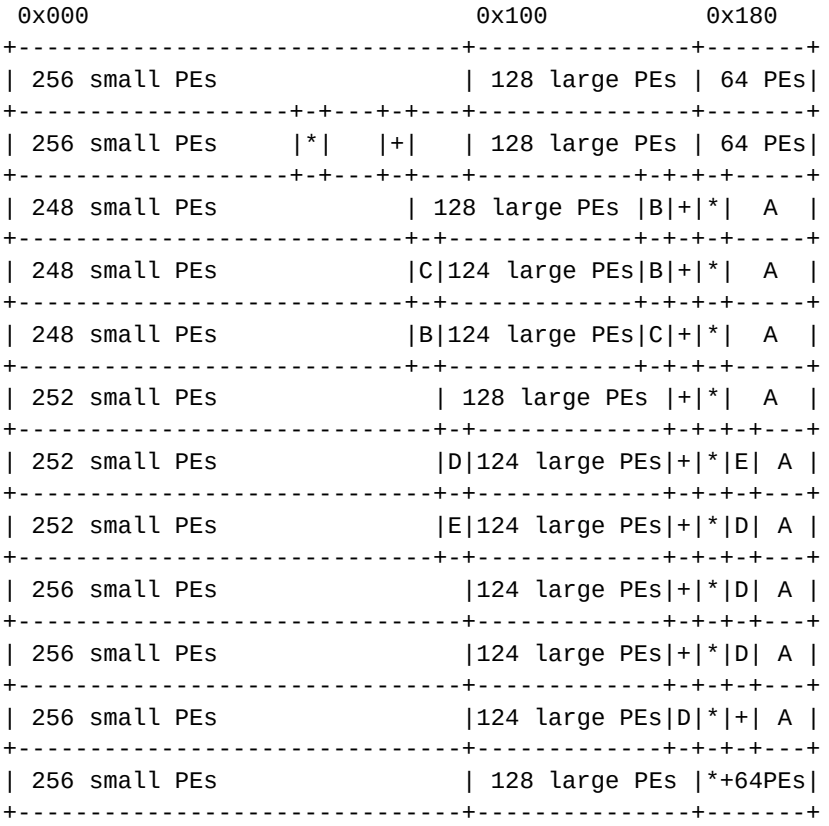
10. Exchange the selected boards. After D and + are exchanged, the configuration is as follows:

```
 0x000                           0x100           0x180
+-------------------------------+-------------+-+-+-+---+
| 256 small PEs                 |124 large PEs|D|*|+| A |
+-------------------------------+-------------+-+-+-+---+
```

After the exchanges, the configuration is 256 small application PEs followed by 128 large application PEs. At the end of the range of applications PEs, 64 PEs still are used for OS, redundant, and command PEs. Two of these 64 PEs are disabled; they are two of the redundant PEs.

```
 0x000                           0x100           0x180
+-------------------------------+--------------+-------+
| 256 small PEs                 | 128 large PEs |+*64PEs|
+-------------------------------+--------------+-------+
```

In summary, following is a diagram of the results of each step in the procedure used in this example:



Cray T3E I/O Controller Nodes [8]

This chapter provides information on the underlying structures that define the configuration of the Cray T3E I/O controller nodes.

8.1 Node Addresses

The Cray T3E mainframe is connected to other mainframe and I/O nodes, such as Multipurpose I/O Node (MPN-1), the IPN-1 IPI-2 I/O Node (IPN-1), the HIPPI channel I/O Node (HPN-1, HPN-2), and the Fibre Channel I/O Node (FCN-1) by means of the GigaRing channel. The I/O controller nodes on the Cray T3E mainframe connect the Cray T3E mainframe to the GigaRing channel. One I/O controller node can connect to one GigaRing only.

The I/O controller nodes of the Cray T3E are referenced by the four physical processing element (PE) numbers that share the same printed circuit board (PCB). Therefore, the I/O controller node on the PCB with physical PE 0, 1, 100, 101 can be referenced with numbers 0, 1, 100, or 101. All four numbers refer to the same I/O Controller node.

Although each I/O controller node can be connected to only one GigaRing channel, many I/O controller nodes can be connected to the same GigaRing channel. A GigaRing channel is identified by an address, which has a 7-bit integer value. Each I/O controller node connected to the GigaRing channel also has a node address, which has a 6-bit integer value. The combination of the GigaRing address and node address form the GigaRing node address, which consists of 7 bits of GigaRing address and 6 bits of node address, as follows:

```
12 11 10 9 8 7 6 5 4 3 2 1 0   Bit Numbers

|    RING NO   | |  NODE NO|
```

The GigaRing node address is not the same as the I/O controller node number. The same I/O controller node has two identifications. From the mainframe, the I/O controller node is referenced by a physical PE number. From the GigaRing channel, the I/O controller node is referenced by the GigaRing node address.

In summary, the I/O controller has the following components:

1. A physical PE number

2. A GigaRing node address
3. A GigaRing number

8.2 Routing Structure

On the Cray T3E mainframe, the I/O controller node, the managing packet server, and the GigaRing channel it is connected to are identified by the following structure (`scxroute_t` in `cs_iosdef.h`):

`ringno`	The GigaRing number to which this I/O controller node is connected.
`io_contr_ppe`	The physical PE number that is used to reference this I/O controller node within the mainframe.
`io_contr_lpe`	The logical PE that references this I/O controller node within the mainframe. (Normally not used; `io_contr_ppe` is used instead.)
`gigaring_node_addr`	The GigaRing node address of this I/O controller node. This address is written onto the I/O controller node by the SWS software. Because the SWS software does not share the same configuration file, you must ensure that the address is the same.
`path_type`	Either primary or alternate. More than one I/O controller node from the same mainframe can be connected to the same GigaRing channel. One is designated as the primary while the others are alternates. The packet server responds to these nodes in a round robin fashion.
`link_type`	Always use `link_scx`.
`intr_ps_name`	The PE name of the packet server that will receive the all interrupts from this I/O controller node.
`send_ps_name`	The PE name of the packet server that will send messages out via this I/O controller node.

`intr_ps_name` and `send_ps_name` are usually the same.

The above describes the identification of the I/O controller node from its mainframe, the GigaRing channel it is connected to, and the address by which

it can be referenced from the GigaRing channel by other nodes, such as the MPN-1 or the HPN-1. It also identifies the packet server, which is the software component that drives the I/O controller node from the mainframe. The above information provides the packet server with the capability of sending messages to clients on a particular GigaRing channel. The packet server needs one of these structures for each I/O node it can send packets to.

8.3 The Packet Server

The I/O controller node is driven on the Cray T3E by the packet server. The packet server can be loaded and executed from any OS PE. An I/O controller node can be addressed from any packet server on any PE. However, only one packet server can receive all the interrupts from an I/O controller node. A packet server can receive interrupts from multiple I/O controller nodes. In summary:

- Only one packet server should be associated with an I/O controller node.
- Only one packet server can receive interrupts from an I/O controller node.
- One packet server can receive interrupts from any number of I/O controllers.
- One packet server can service multiple rings.
- One packet server can support at most one local and seven remote disk servers.

  Note that if you are using the alternate path feature in your disk configuration(s), and if you are using different packet servers for the primary and alternate paths, you will be using up some of the allowed remote disk server entries to support the alternate path. Similarly, if you are using different disk servers for the primary and alternate paths, you will be using up some of the allowed remote disk server entries to support the alternate path.

The packet server is identified by the name of the OS PE where the packet server is loading and executing.

8.4 Packet Server Routing

The packet server provides reliable messaging capability to the `disk`, `tape`, `tty`, and `netdev` servers running on OS PEs. Since more than one I/O controller node can be connected to the same GigaRing channel and each of

these nodes can be managed by a different packet server, you need to be able to link these packet servers to the other servers (`disk`, `tape`, etc.) that are using them.

For example, if I/O controller nodes 0, 4, and 8 are connected to GigaRing 5 and the packet server on `ospe_a` manages I/O controller nodes 0 and 4 while the packet server on `ospe_b` manages I/O controller node 8, you need to allocate one of these packet servers to servers that need to send messages out to GigaRing 5.

The following information and example show how you might allocate the server if the disk server on `ospe_a` wants to send to GigaRing 5.

The structure `server_route_t` in `cs_iosdef.h` defines the configuration between the calling server (disk) to the packet server for a particular GigaRing. This structure contains the following:

`server_name`	The calling server name; for example, `disk` or `tape`.
`server_pe_name`	The name of the PE on which the calling server is loaded. The PE name is needed because there can be more than one disk server on a mainframe.
`ringno`	The GigaRing number of the client to which the calling server wants to send the message.
`ps_pe_name`	The name of the PE on which the packet server is loaded.

If you provide the following information, the disk server in `ospe_a` will call the packet server in `ospe_b` to send messages to GigaRing 5. In this example, the packet server in `ospe_b` will use I/O controller node 8 to get to GigaRing 5:

```
server_routes[0] {
    server_name = "disk";
    server_pe_name = "ospe_a";
    ringno = 5;
    ps_pe_name = "ospe_b";
}
```

8.5 Configuration Checklist

You need the following information to configure an I/O controller node and packet server, and to configure servers such as disk and network servers to the packet server. These questions should provide you with enough information to

configure the `scxroute_t` entry. Answer them for all connected I/O controller nodes:

1. Which printed circuit board (PCB) contains the I/O controller node? What are the physical PE numbers of these PEs? Any of one of these physical PE numbers can be used to identify this I/O controller node.
2. What is the GigaRing number to which this I/O controller node is connected? This GigaRing number is used by the SWS to initialize the ring.
3. What is the GigaRing node address of this I/O controller node? This address is used by the SWS software to initialize this node and ring.
4. Which packet server will be dedicated to receive all the interrupts from this I/O controller node? You need to know the PE name on which this packet server is loaded (for example, `ospe_a`, `ospe_b`, etc.).
5. Which packet server will be sending message packets out to this ring using this I/O controller node?

Normally, the answers to questions 4 and 5 are the same.

The following questions should provide enough information to configure the `server_route_t` entries. Answer them for all servers.

1. What servers will be sending message packets and will require the services of the packet Server? The `disk`, `tape`, `tty`, and `netdev` servers are the only servers that require service from the packet server.
2. What are the PE names where these servers are loaded (for example, `ospe_a`, `ospe_b`, etc.).
3. For a particular server to a particular GigaRing channel, what configured packet server do you want to use? You need the PE name on which this packet server is loaded. Make sure that this packet server manages at least one I/O controller node connected to this GigaRing channel.

Disk Device and File System Configuration [9]

In a UNICOS/mk system, configuration of all disk devices except `root` can be performed through the `mknod`(8) command. This method of disk configuration is the same method that is used in the UNICOS operating system.

Use of the `mknod`(8) command to configure disk devices is described in *UNICOS/mk General Administration*.

> **Note:** When you add additional file systems to your system, you may need to increase the size of the system buffer cache. You do this by increasing the value of the `os {nbuf =` *xx*`; }` parameter in the section of the configuration file that defines the mainframe parameters.

The description of a disk device has the following components:

- The primary and alternate I/O path and unit number of the disk
- The PE name of the disk server managing this disk device
- The type of disk

This chapter describes each of these components.

This chapter also provides a procedural overview of disk configuration and miscellaneous disk configuration information.

9.1 I/O Path and Unit Number

Disk devices on a GigaRing channel are accessed by the disk server with the I/O path and unit number. The I/O path of a disk device consists of the following:

- The GigaRing number the disk device is on. This is a 7-bit value.
- The node number of the single-purpose node (SPN) or multipurpose node (MPN) to which the disk is connected. The node address is a 6–bit value.
- The controller number and the unit number of the device. These numbers depend on the type of disk device and the node to which the disk is connected, as described in the following sections.

In summary, the I/O path of a disk consists of a GigaRing number, a node number, a controller number, and a unit number. These four values uniquely identify a disk device.

In the UNICOS/mk configuration file, the GigaRing number of a device is defined with the `iopath.ioc` member of the device definition, the node number is defined with the `iopath.iop` member, and the channel is defined with the `iopath.channel` member.

9.1.1 Disk Devices on the MPN-1

For disk devices connected to the MPN-1 (SCSI disks):

- The SCSI controller number is in the range 0 through 8.
- The unit number is in the range 0 through 15.

9.1.2 Disk Devices on the IPN-1

For disk devices connected to the IPN-1 (IPI-2 disks):

- The controller number is in the range 0 through 4. For array devices, the channel number is always 0.
- The unit number is in the range 0 through 7.

9.1.3 Disk Devices on the FCN-1

For disk devices connected to the FCN-1 (Fibre SCSI disks):

- The controller number is in the range 0 through 4.
- The unit field is defined as follows:
  - Bits 0 through 7: FC Loop_ID number of disk unit, as described below.
  - Bits 8 through 15: Logical Unit Number (LUN). Currently this number must be 0.

The FC Loop_ID is determined as follows:

The Disk Subsystem Fibre Channel (DSF-1) assigns all the enclosed disk units hard addresses, the lower 4 bits of the Loop_ID. Each DSF-1 can be assigned an address, on their I/O cards, bits 4, 5, and 6 of the Loop_ID. Bit 7 is not used and must be zero. There are ten drives in a DSF-1, five in front, five in back;

they plug into both sides of a printed circuit (PC) board, called the mid-plane. Bit 3 of the Loop_ID identifies whether the drive is in the front or back. Bits 0, 1, and 2 identify the disk units. This yields a total of 80 disk units per FC Loop; the ANSI standard allows 126.

This arrangement is illustrated in the following diagram:

```
7  6  5  4  3  2  1  0
               +-----+  drive (of five)

            +           front/back (0 = front)

+------+                DSF-1 address.
```

Loop_IDs (in octal):

001–005	Front drives of DSF-1 0
011–015	Back drives of DSF-1 0
0161–0165	Front drives of DSF-1 7
0171–0175	Back drives of DSF-1 7

Disk arrays must use the front or back 5 consecutive drives of a DSF-1. For example, arrays are formed from units 001 through 005, 011 through through 015; the general format is 0?1 through 0?5.

RAID-3 spindles must have the same value for bits 3-7. The parity spindle is Loop_ID nn1 (the lowest numbered). Non-RAID devices can be on the same Loop.

RAID-5 spindles must start with Loop_ID nn1. All RAID-5 devices on the same Loop must be the same width. Neither RAID-3 nor non-RAID devices may be on the same Loop with RAID-5 disks.

The RAID-3 array and spindle addressing convention is as follows:

iopath.21	This would indicate an array reference.
iopath.21–*x*	This would indicate a spindle *x* reference.

The first unit of the five drives in the array is used to reference the disk array. The first unit and a spindle is used to reference the spindle within the disk

array. For example, the following command references the disk array, units 021 through 025:

```
xdms -a reconstruct 04040.21
```

The following command references spindle 2 (unit 23):

```
xdms -a disable 04040.21-2
```

For further information on configuring disk devices on the FCN-1, see Section 9.6. For information on configuring disk arrays, see Section 9.7, page 139.

9.2 Disk Server Configuration

The UNICOS/mk system allows you to spread the disk device i/o workload across multiple OS PEs by configuring multiple disk servers. Each disk device is associated with one disk server, based on information in the configuration file. There are two ways to assign a disk device to a disk server:

- Include a `phys_stor[]` configuration specification for the disk device in the configuration file used to boot the mainframe. The `phys_stor` entry includes a `disk_pe` definition, which names the OS PE containing the disk server assigned to the disk device.
- Include `diskdriver[]` structures in the parameter file. The diskdriver entry includes a `disk_pe` parameter to specify an OS PE containing a disk server, and `ion[]` parameters that specify ring and node numbers. All disk devices on the specified rings and nodes are served by the `disk_pe` associated with this diskdriver entry, unless any device is explicitly associated with a different disk server OS PE in a `phys_stor` definition (an explicity `phys_stor disk_pe` definition will override the `diskdriver[]` association for a single disk device).

9.3 Supported Disk Types

The following disk devices are supported on the MPN-1:

- DD-314
- DD-318

The following disk devices are supported on the FCN-1:

- DD-304

- DD-308
- RAID-3
- RAID-5

The following disk devices are supported on the IPN-1:

- DD-60 and DA-60
- DD-62 and DA-62
- DD-301 and DA-301
- DD-302 and DA-302

9.4 Changing File System Configuration

The Cray T3E system is preconfigured with a basic set of file systems. In order to have a fully functional system, you must create user file systems, resize existing file systems as appropriate for your needs, and consider file system layout issues.

Note: When you add additional file systems to your system, you may need to increase the size of the system buffer cache. You do this by increasing the value of the `os { nbuf =` *xxx*`; }` parameter in the section of the configuration file that defines the mainframe parameters.

The Cray T3E system is preconfigured with the following file systems:

- `/` (the root file system)
- `/usr`
- `/opt`
- `/tmp`
- Swap partition (shared with boot PE dump partition)
- `/dump` (dump file system)

Caution: UNICOS/mk upgrade releases will perform a full replacement of the `/usr` file system. If you create subdirectories in `/usr` (such as `/usr/spool` or `/usr/adm`) that you want to be preserved after an upgrade, you must ensure that they are configured as separate file systems.

You will need to create the other file systems your users need. You should configure your file systems to minimize contention. Many sites create the following additional file systems:

- User home directories (sometimes called `/home` or `/w`)
- `/usr/tmp`
- `/usr/adm`
- `/usr/spool`

9.4.1 Configuring a File System

File system configuration is defined in the disk nodes, which you configure on the Cray T3E system using the `mknod(8)` command. All devices except the root device can be configured with `mknod(8)`.

In UNICOS systems, `swap` devices were configured in the configuration file. In UNICOS/mk systems, you can configure a swap device with the `mknod(8)` command. However, you still need to provide the minor number of the node on the root file system that defines the swap device in the configuration file.

You can configure file systems in one of two ways:

- Define your file system configuration in the configuration file. Boot the system in single-user mode and run the `fconfig(8)` command to generate a script which includes the `mknod(8)` commands to create the file system nodes.
- Define `root` in the configuration file and create all other devices and file systems directly using the `mknod(8)` command. Information on using the `mknod(8)` command is provided in *UNICOS/mk General Administration*.

Whichever method you use to configure your file systems, you should perform the following procedure before creating your disk nodes:

1. Verify that all configuration changes that you will be making are reflected in the topology file (`/opt/config/topology`). For example, all rings and I/O nodes referred to in the configuration file must also be described in the topology file so that the I/O nodes are booted and the rings are initialized by the SWS when `bootsys`(8) is run.

   For a full description of the procedure required to configure PEs and GigaRing nodes, see Chapter 6, page 81.

2. Check the maximum device values in the mainframe and `diskdriver` definitions in the configuration file, and increase the values if necessary:

```
mf[0] {
       ...

       drivers {
                diskdriver[0] {
                        pddmax   = 32;
                        xddmax   = 256;
                        pddslmax = 256;
                        rddslmax = 4;
                        xddslmax = 256;
                }
                ...
        }
        ...
}
```

If you choose to define your file system configuration in the configuration file, use the following procedure:

1. On the SWS, edit the configuration file to make the following changes:

   a. Define the physical disk (the `pdisk` structure). There must be one `pdisk` entry for each physical disk device. Create a new entry (or use an unused entry); for example, 50. Here is a sample entry:

```
mf[0] {
      ...

      disk_info {
              ...

              phys_stor[50] {
                       name = "mooII.s40"; disk_pe = "ospe_b";
                       ptype = d_disk; stype = d_x500;
                       iopath { ioc = 0; iop = 1; chan = 0; }
                       unit = 0;
              }
              ...
      }
      ...
}
```

The following fields are of interest:

Field	Description
`name`	Name of device. You can select anything, but the naming convention is `d`*type*`.s`*xxx* (for example, `dscsi.s123`).
`iopath.ioc`	Ring number. This is the number of the GigaRing channel that the device is on. (For example, use 1 on systems with 1 GigaRing channel.)
`iopath.iop`	Node number. This is the number of the MPN (for example, 7).
`iopath.chan`	Controller or slot number. This is the slot holding the SCSI controller.
`unit`	Unit number. This is the SCSI ID number for the physical disk.
`disk_pe`	PE name. This is the name of the OS PE that the disk driver runs on. If you do not know which PE to specify for this field, see Section 9.4.2, page 133, for the procedure to determine the PE name. If you don't specify an OS PE for the disk driver, it defaults to the OS PE specified for `diskdriver[0]` in the configuration file.

b. Define the `xdd` slices (the physical partitions for the disk), as in the following example:

```
mf[0] {
      ...

      disk_info {
              ...

              xdd[0] {
                      minor = 1; name = "mooII.s40";
                      start = 0; length = 12000; length_unit= blocks;
                      pstor = &mf[0].disk_info.phys_stor[50];
              }
```

```
                ...
        }
        ...
}
```

Note: Keep like devices together (group `xdd` entries by `phys_stor` number) for convenience. (Sequential is good for organization.) For the minor device number, use anything unique.

c. Create the `dsk` (the logical disk device, or file system partition), as in the following example:

```
mf[0] {
        ...

        disk_info {
                ...

                dsk[0] {
                        minor = 1; name = "usr.spool"; nslices = 1;
                        member[0] {
                                mtype = m_xdd;
                                mstor { phys = &mf[0].dev.xdd[0]; }
                        }
                }
                ...
        }
        ...
}
```

Every `xdd` must be associated with an `ldd`, which can have one or more slices. You need one `ldd` member entry (`mstor.phy` and `mtype`) for each physical partition (`xdd`). The example structure in this step shows a logical device with one physical slice.

2. Check for overlaps in the disk definition before booting the mainframe. The SWS `csv`(8) command may catch some overlaps, but there is currently no foolproof tool to do this task. (It is possible that running `mkfs`(8) will detect overlaps.)

3. Reboot the Cray T3E system to single-user mode.

4. On the Cray T3E system in single user mode, run the `fconfig`(8) command to generate a node script. The node script should be a series of `rm`(1) and `mknod`(8) commands.

   ```
   fconfig -dr > /etc/nodes.sh
   ```

   You may see error messages indicating that there are duplicate device names or device names that are too long. Edit the configuration file to correct these errors, reboot the system to single-user mode, and run the `fconfig`(8) command again to correct these errors.

5. If special diagnostic nodes for IPN-1 devices are located in the configuration file and have a `flags` field set to 0137, the `nodes.sh` script will include commands to generate these nodes in the `/dev/ddd` directory.

6. Run the `nodes.sh` script generated by the `fconfig`(8) command.

7. Run the `fsck`(8) command on the root file system:

   ```
   /etc/fsck /dev/dsk/root
   ```

8. Run the `sync` command.

   Note: It is very important that you run the `sync`(8) command before you reboot the system after creating the new nodes on your system.

9. Shutdown and reboot the system.

After you have created your device nodes with the `mknod`(8) command, either manually or by running the script generated by the `fconfig`(8) command, you can perform the following procedure for each file system:

1. On the Cray T3E system, create the file system with `mkfs`(8).

2. On the Cray T3E system, verify the file system with `fsck`(8).

3. On the Cray T3E system, mount the file system with `mount`(8). If you want the file system to be automounted during a boot to multiuser mode, add the file system to `/etc/fstab`.

4. On the Cray T3E system, test the new file system using this procedure:

   a. Run `df`(8) to ensure that the new file system appears and looks OK (starting block, size, and so forth).

   b. Copy a file to the new file system.

9.4.2 Determining the Disk PE

You can use the following procedure to determine this name of the OS PE that a disk server runs on.

1. Examine the `pe[*]` entries in the configuration file. Look for one that has a PE type of `PE_TYPE_OS`, as in the following example, and find the associated actor list index (the `actor_list_index` field) for the PE:

```
pe[18] {
       flags = 0;
       pe_type = PE_TYPE_OS;
       actor_list_index = 4;
       pe_name = "ospe_b";
}
```

2. Use the associated actor list index to view the entry in the `pe_actors[*]` array.

```
/* pe_actors index 4 -- OS PE -- two-OS-PE system -- second OS PE */
pe_actors[4] {
        actor_name[0] = "kernel";
        actor_name[1] = "em";
        actor_name[2] = "PM";
        actor_name[3] = "packet";
        actor_name[4] = "disk";
        actor_name[5] = "netdev";
        actor_name[6] = "file";
        actor_name[7] = "socket";
        actor_name[8] = "sio";
        actor_name[9] = "tape";
}
```

3. If one of the entries in the `pe_actors[*] { actor_name = "`*xxx*`"; }` array says `disk`, you have found an OS PE that is running a disk server:

```
pe_actors[4] {
        ...

        actor_name[4] = "disk";
        ...
}
```

4. Finally, to find the name of a PE that uses this set of actors, return to the OS PE (in this example, PE 18), and look for the PE name:

```
pe[18] {
       ...

       pe_name = "ospe_b";
}
```

So, in this example, we can specify the PE name as follows:

```
phys_stor[50] {
        disk_pe = "ospe_b";
        ...
}
```

9.5 Configuring Dump Files

This section describes the files, directories, and file systems that must be configured for UNICOS/mk dumps to work correctly. It also describes a procedure to estimate how much space you will need for system dumps.

9.5.1 Configuring a Dump Device

You can configure a device for the Cray T3E system to use as a raw dump device instead of the `/opt/CYRIdump` directory on the SWS. For information on creating a postmortem dump, see the "Crash and Dump Analysis" chapter of *UNICOS/mk General Administration*.

The following example shows excerpts from a configuration file that configures a boot and a dump device.

The following section of the configuration file defines the disk on an MPN where the boot and dump slices will be found:

```
phys_stor[0] {
        name = "dscsi.s40"; disk_pe = "ospe_b";
        ptype = d_disk; stype = d_x500;
        iopath { ioc = 0; iop = 1; chan = 4; }
        unit = 0;
}
```

The xdd definitions in the configuration file define the starting block offset and length of the slices on an MPN disk. These need to match the /dev/xdd nodes on the mainframe. The following shows an example of xdd definitions:

```
xdd[93] {
        minor = 7; name = "boot";
        start = 1000000; length = 10000; length_unit = blocks;
        pstor = &mf[0].disk_info.phys_stor[0];
}

xdd[94] {
        minor = 8; name = "dump";
        start = 1010000; length = 50000; length_unit = blocks;
        pstor = &mf[0].disk_info.phys_stor[0];
}
```

The dsk entries in the configuration file define the logical mapping to the xdd slice (/dev/dsk node to /dev/xdd node on the mainframe). The following shows an example of dsk entries:

```
/* logical devices */

dsk[14] {
        minor = 7; name = "boot"; nslices = 1;
        member[0] {
                mtype = m_xdd;
                mstor { phys = &mf[0].dev.xdd[93]; }
        }
}

dsk[15] {
        minor = 8; name = "dump"; nslices = 1;
        member[0] {
                mtype = m_xdd;
                mstor { phys = &mf[0].dev.xdd[94]; }
        }
}
```

The boot and dump devices are system devices, and are defined in the configuration file with the system device definitions, as in the following example:

```
/* System device definitions */

archdev { name = "boot"; minor = 7; }
dumpdev { name = "dump"; minor = 8; }
```

9.5.2 Tips for Configuring Dump File Systems

Consider the following recommendations when configuring dump file systems:

- Whenever possible, the dump file system should be striped across multiple disks. Striping the dump file system across multiple disks greatly decreases the amount of time it takes to dump the PEs to disk. If there are not enough disks on a single ION (FCN or MPN), the dump file system can be striped across multiple IONs.

  Some sites running the UNICOS/mk operating system are configured with dump file systems consisting of as many as 60 slices striped across disks spanning up to 4 MPNs or FCNs. This configuration allows efficient use of the hardware and permits fast full Cray T3E dumps.

- On rare occasions, the system may hang when dumping the mainframe PEs to disk. This happens when the `mount`(8) command hangs while mounting the dump file system (usually `/dev/dsk/dumps` is mounted on `/dumps`). The reason the `mount`(8) command hangs is that the mount operation is waiting for buffers when the file system to be mounted requires more than 2000 buffers for the file system block map.

  In mainframe dumps, the value given in the configuration file for system buffers (i.e. in `mf[0].os.nbuf`) is always ignored and a value of 2000 is used instead. This is done to avoid running out of memory, since resources are limited on the boot PE, the only PE that gets restarted on a system dump.

  For dump file systems that require large block maps, you should remake the file system with a larger secondary allocation unit. This will reduce both the number of map blocks and the number of buffers required for the dump file system block map.

  By default, both primary and secondary allocations are always in single blocks. If possible, you should remake the dump file system so that the secondary is increased by a factor of 4. This makes the map to block ration

1:4, where each entry in the block map refers to 4 blocks in the secondary allocation.

Note that it is a normal sleep condition for the system to hang on mount under the above circumstances. It is also possible, in certain cases, that the buffers will be flushed and that the waiting mount will complete normally.

9.5.3 UNICOS/mk Dump File System (/dumps)

The UNICOS/mk dump file system, /dumps, holds the UNICOS/mk portion of the system dump. As in the UNICOS system, the dump file system consists of time-stamped directories containing the individual portions of a complete system dump.

By default, this file system is called /dev/dsk/dumps, and is mounted as /dumps. (In the UNICOS system, it was called /core). The /etc/config/rcoptions file contains the parameters that define this file system.

Note: If the dump file system is configured as part of the /dev/dsk file system rather than as a separate file system, you must ensure that there is enough space for an entire dump, and that the file system is not destroyed with mkfs(8) before the dump has been saved in an alternate location.

9.5.4 The /etc/config/rcoptions File

As in the UNICOS system, the UNICOS/mk file /etc/config/rcoptions specifies the configuration parameters for the dump file system, such as the location, mount point, and optional dump directory (if one exists outside the dump file system).

You can edit the rcoptions file with the ConfigTool utility, as decribed in Chapter 4, page 41.

The following parameters are of particular importance:

Parameter	Description
DUMPCMPAPP	Specifies that the cpdmp(8) compression processes run by coredd(8) or sysdumpmf(8) will be run on application PEs using the mpprun(1) single PE application option (-a); enable this parameter only if your site configures one or more application PEs to run single PE applications. This parameter is disabled by default.

`DUMPDIR` Specifies the directory into which the dumps are to be written; this parameter is set to " " (null) by default.

`DUMPFS` Specifies dump file system (in `/dev/dsk`) to which the dump will be copied; `core` is the default dump file system.

`DUMPMPT` Specifies the mount point for the dump file system; `/mnt` is the default mount point.

`DUMPNPROC` Specifies number of concurrent `cpdmp`(8) compression processes (`DUMPNPROC`) that will be run when compressing a Cray T3E system dump via `coredd`(8) or `sysdumpmf`(8). By default, four (4) processes will be run in parallel during the compression. Values from `1` to `4096` are possible, but site disk configuration (affecting the speed of the dump file system) may make values greater than `16` superfluous.

9.5.5 The `/.netrc` File

This UNICOS/mk file `/.netrc` permits automatic `ftp`(1) access for the root user from the Cray T3E system to the SWS. This file must contain the machine name of the SWS, the SWS account name used to run dumps (`crayops` by default), and the password for that account, as shown in the following example:

```
machine sn0000-sws login crayops password initial1
```

Note: Sites that choose not to allow this FTP access should be aware that the dump procedure will not be fully automatic. The administrator will have to review startup messages and the log file `/etc/dump.log` to determine which portions of the dump and system archive still reside on the SWS, then manually transfer these files to the correct dump directory on the UNICOS/mk dump file system.

9.5.6 The `/etc/dump.log` File

The UNICOS/mk file `/etc/dump.log` stores dump log information.

9.6 Configuring Multiple FCN Devices

When you configure multiple FCN devices on GigaRing channels in UNICOS/mk systems, you must account for various restrictions regarding the assignment of disk servers. Some of these restrictions are general restrictions on mirrored and striped devices, which are described in the "Restrictions on striped and mirrored logical devices" section of the "File System Planning" chapter of *UNICOS/mk General Administration*.

When you configure FCN devices, you must take the following into account:

- It is recommended that you use one disk server OS PE for every 2 FCN devices. If you use a single disk server for more than 2 FCN devices, the OS PE can run out of memory and you will be unable to boot the system.
- If you stripe across FCN devices, the devices must be served by the same disk server.
- If you mirror across FCN devices, the devices must be served by the same disk server.

These restrictions have the following implications:

- If one disk in a disk array is part of a striped group, the entire disk array must use the same disk server on the same OS PE as the rest of the devices in the striped group.
- Similarly, if one disk in a disk array is part of a mirrored group, all the disks in the array must use the same disk server as the disks in the mirrored group.

You must plan out the arrangement of your disk arrays, striped disks, and mirrored disks with great care. If you do not observe these restrictions when configuring your system, the problem will manifest itself in ways that are difficult to analyze. For example, you may try to execute a `mkfs`(8) command and the system will hang if the configuration violates these restrictions.

9.7 Configuring Disk Arrays

This section contains information on how to configure DD-308 devices on FCNs in RAID-3. The examples in this section use the `mknod`(8) command to configure disk devices. For information on using the `mknod`(8) command to configure disk devices, see *UNICOS/mk General Administration*.

9.7.1 Installing an Array

Perform the following steps to install and configure a disk array:

- Copy any data to be saved to another media
- Initialize the device as a RAID-3 array
- Write data to the parity drive
- Modify configuration information

After you perform these steps, the array is ready for I/O. These steps are described in detail below.

1. Copy any data to be saved to another media.

   It is not possible to move a file system on 4 single drives onto an array of 4+1 drives without first dumping, then restoring the data.

   There are two factors that may affect the restore of the data:

   - If there are many small files, the restored data could take more space. This is because the minimum sector size is four blocks, so small files of one to three blocks may take more space when they are restored.
   - For extended files, the space allocation may be more efficient after the restore and take less space.

   Unless the existing file system is nearly full, it is unlikely that it will not restore.

   When dumping and restoring a file system on single spindles to a file system on an array (or more than one array), allow for the fact that a RAID device holds only as much data as four single devices. Because of this, you must plan carefully when migrating filesystems from single disks to RAID disks.

2. Initialize the device as a RAID-3 array:

   ```
   xdms -a init -m 35 /dev/xdd/name
   ```

   or

   ```
   xdms -a init -m 35 xdms -a init -m 35 02010.11 # 011 (Octal)
   ```

   See the `xdms`(8) man page for details.

The syntax `-m 35` means raid 3, 5 units. The 35 does not indicate a mask, but a hex code identifying the raid type (mode) to be initialized.

3. Write data to parity drive.

   ```
   xdms -a scrub /dev/xdd/name
   ```

   or

   ```
   xdms -a scrub 02010.11
   ```

   Executing a `scrub` of a DA-308 takes approximately thirty minutes.

   If the scrub fails or is interrupted with Control-C, you will see errors when you execute `mkfs`(8) on the file system

4. Modify configuration information.

   If the DD-308 disks were previously used as single spindles, all `xdd` and `dev/dsk` entries should be removed and the file systems unmounted before configuring the disks as RAID-3 devices.

5. Create the `xdd` entry for the RAID-3 device with the `mknod`(8) command:

   ```
   /etc/mknod /dev/xdd/name c 33 minor 4 pripath 0 2340000 0 altpath unit
   ```

   The *dtype* field needs to be 4 (to indicate a RAID-3 device); the 4 specifies that there are four 512-word blocks per sector. *Loop_ID* is sometimes used instead of *unit* number. Only one `/dev/xdd/`*XXXX* node is needed to represent an entire array.

   The following example shows the output from a `ddstat -m disk*` command on an array:

   ```
   /etc/mknod disk1 c 33 39 4 02010 0 2340000 0 03010 01 0
   /etc/mknod disk2 c 33 40 4 02010 0 2340000 0 03010 011 0
   /etc/mknod disk3 c 33 41 4 02020 0 2340000 0 03020 01 0
   /etc/mknod disk4 c 33 42 4 02030 0 2340000 0 03030 01 0
   /etc/mknod disk5 c 33 43 4 02030 0 2340000 0 03030 011 0
   ```

   The last line in the above example represents a RAID-3 slice on ring 2, node 3, FCN channel 0, involving units 011 through 015.

Configure the /dev/dsk entries:

```
/etc/mknod disk1 b 34 39 0 0 /dev/xdd/disk1
/etc/mknod disk2 b 34 40 0 0 /dev/xdd/disk2
/etc/mknod disk3 b 34 41 0 0 /dev/xdd/disk3
/etc/mknod disk4 b 34 42 0 0 /dev/xdd/disk4
/etc/mknod disk5 b 34 43 0 0 /dev/xdd/disk5
```

The following example shows the output from a ddstat -m * command:

```
/etc/mknod disk1 b 34 39 0 0 /dev/xdd/disk1
        /etc/mknod /dev/xdd/disk1 c 33 39 4 02010 0 2340000 0 03010 01 0
/etc/mknod disk2 b 34 40 0 0 /dev/xdd/disk2
        /etc/mknod /dev/xdd/disk2 c 33 40 4 02010 0 2340000 0 03010 011 0
/etc/mknod disk3 b 34 41 0 0 /dev/xdd/disk3
        /etc/mknod /dev/xdd/disk3 c 33 41 4 02020 0 2340000 0 03020 01 0
/etc/mknod disk4 b 34 42 0 0 /dev/xdd/disk4
        /etc/mknod /dev/xdd/disk4 c 33 42 4 02030 0 2340000 0 03030 01 0
/etc/mknod disk5 b 34 43 0 0 /dev/xdd/disk5
        /etc/mknod /dev/xdd/disk5 c 33 43 4 02030 0 2340000 0 03030 011 0
```

After performing the above steps, the array is ready for I/O:

```
# /etc/mkfs -q -A96 /dev/dsk/disk1

/etc/mkfs: *** NC1FS filesystem initialized on /dev/dsk/disk1 ***
        *** Lower security level = 0   Upper security level = 0
        *** Valid security compartments = 0
                none
        *** Big file: 32768 bytes  big allocation unit: 96 blocks
        *** Allocation strategy: Round robin all files(rrf)
        *** 1 partitions / 9360000 total blocks / 9351704 free
        *** 131072 total inodes / 131068 free
        *** 1 primary partitions / 4 blocks per alloc. unit
        *** File system partitions:
                part 0: primary blocks 0 - 9359999  on device disk1
        *** Panic on error option selected
```

9.7.2 Replacing a Failing Spindle

Use the following procedure to replace a failing spindle when unrecovered errors are occurring.

1. Determine the failing spindle from the failing spindle mask provided in the `errpt` output or system console log. See Table 6 for more information.

Table 6. Spindle to Unit Number Mapping

Unit	Spindle	Spindle mask
0?1	4	020 (parity)
0?2	3	010
0?3	2	004
0?4	1	002
0?5	0	001

2. Disable the spindle:

   `xdms -a disable` *iopath.unit-spindle*

   For example:

   ```
   xdms -a disable 02010.1-3 # disable spindle 3
   ```

   This step, disabling the spindle, may take place automatically, depending on the type of errors encountered.

3. Spin down the spindle:

   `xdms -a spindown` *iopath.unit-spindle*

   If the I/O to the array become hung (a process in wait I/O) as a result of the `spindown` command, perform the following steps:

   a. Check that the hung I/O is through the FCN associated with the I/O path on `spindown` command.

   b. Wait for a `request timeout` message to appear on the UNICOS/mk console. This message will take about three to four minutes to appear.

   c. Reboot the FCN.

   d. Save the FCN dump.

If you try to spin down a drive that is not disabled, the next I/O to that array/spindle will cause the array to spin up as part of normal error recovery.

4. Replace the failing spindle.

   Note: Pulling or insertion of a spindle will cause the FCN software to re-initialize the DSF-1. This DSF-1 initialization may take a few minutes and will affect not just the DSF-1 containing the pulled/pushed spindle (which will suspend I/O to other drives in the DSF-1), but will affect other DSF-1s on the same daisy-chained channel. Some error/console message may appear at this time.

   Before reconstructing the array, check that the serial number of the drive/unit (on the I/O path) is correct:

   `xdms -a info` *iopath.unit-spindle*

   To get around the problem of not having access to the serial numbers of raid spindles, use `xdms -a disk` to display all devices of the FCN.

   The `xdms`(8) command will abort with the message `Raid device selection does not conform to access convention!` if a unit other than the parity drive (0?1) is given. If the parity drive unit number is given, information is provided about the array, not just the parity spindle.

5. Reconstruct the spindle.

   The replacement drive that is to be used to reconstruct the full 4+1 array does not need to be initialized.

   Execute the following command:

   `xdms -a reconstruct` *iopath.unit*

Note: A reconstruct with alternate path requires that the `xdms reconstruct` command is run on the active path to the RAID device. Use the `sdstat`(8) command output to determine the current active path.

If a reconstruct is in progress, and the RAID device switches to alternate path, or the FCN receives a `PEER DOWN` from the mainframe, the current reconstruct will abort. If will be necessary to restart the recontruct from the now-active alternate path (again, you should verify the active path with the `sdstat`(8) command). If the reconstruct is restarted from the alternate path, do not configure the RAID device(s) being reconstructed back to their primary path until all reconstructs complete.

A reconstruct can take 1.5 hours, minimum, depending on other I/O to the same disk array or other I/O to the same DSF-1. Some reconstructs have taken over 10 hours to complete, when running heavy I/O to the same array (for example, by using `fstest`(8)).

Only one reconstruct per loop is recommended. Any other reconstruct on the same loop will be in competition for the channel loop. One of the reconstructs dominates and completes first. Multiple concurrent reconstructs to different loops on the same FCN are OK.

A message appears on the console (and in the `ion_syslog.info`) when the reconstruct is complete:

```
10/16/97 17:39:17 NOTICE sdisk_admin_r3.c line 774
Array Reconstruction is complete on FC Loop 0 Target 1
```

9.7.3 Converting RAID Members to Single Spindles

You should not initialize RAID-3 members to single spindles when one spindle of the array is bad or missing. Initialization of a spindle after a failed `init` of a bad or missing spindle can result in an FCN panic, such as `tBoot: [SPN-1] EMERGENCY: bootConfig.c line 1338 ; EXCEPTION PANIC:`. Use one of the following checks to avoid this situation:

- Use `xdms`(8) to verify that all spindles on a specific I/O path exist and are accessible:

  `xdms -a disk` *iopath*

  If `xdms`(8) cannot provide information on a spindle, do not try to initialize that spindle (`xdms -a init`).

- Replace the bad spindle, checking the new spindle with `xdms -a info` before trying the initialization. If a replacement spindle is not available, then the bad spindle can be moved the rightmost slot of the DSF-1 (unit 0?5 or spindle 0) to avoid the problem.

  After initializing an array with a disabled spindle back to a single spindle using `xdms -a init -m 1`, any subsequent `xdms`(8) command (even a simple `xdms -a info` command) will fail on the open and cause one or more of the spindles to revert to array mode. To avoid this problem, reboot the FCN immediately after the single spindle initialization before referencing or opening any of the former array members. If the DSF-1 is connected to another FCN, there may be I/O errors when the FCN resets the DSF-1 after its reboot.

The spindle can be initialized in a slot that would normally be part of an array, although this is not required to perform a reconstruct.

The following example initializes one member of a RAID to a single spindle:

```
# ./xdms -ainit -m 1 4044.21

4044.21 appears to be a member of a RAID-3S 4+1    .
To init JUST THIS MEMBER to Single Spindle enter "y"
To init ALL RAID MEMBERS to Single Spindle enter "a"

Please be sure all activity to device is idled before continuing with init.
Continue with Init to Single Spindle now? (y, a, or n)
y
xdms: Initialized iopath 4044.21 as Single Spindle
```

Note: This command is different from the `xdms`(8) command to init the full array, which uses `-m 35`.

To turn a disk array back into five single disks, initialize each spindle of the array or follow the example below.

The following example initializes all RAID members to a single spindle:

```
# ./xdms -ainit -m 1 4044.21

4044.21 appears to be a member of a RAID-3S 4+1    .
To init JUST THIS MEMBER to Single Spindle enter "y"
To init ALL RAID MEMBERS to Single Spindle enter "a"

Please be sure all activity to device is idled before continuing with init.
Continue with Init to Single Spindle now? (y, a, or n)
a
xdms: Initialized iopath 4044 unit 21
xdms: Initialized iopath 4044 unit 22
xdms: Initialized iopath 4044 unit 23
xdms: Initialized iopath 4044 unit 24
xdms: Initialized iopath 4044 unit 25
```

Note: If the `-z` option is used with the `-a init` action and the device mode is a `-m 1`, only the RAID member specified will be initialized. The net affect of the `-z` option is the same as in previous example.

9.7.4 Software Limitations

The following limitations are in effect when configuring and restoring disk arrays.

- Executing a Control-C after issuing a `xdms -a reconstruct` command does not halt the reconstruct. After the initial reconstruct request, the FCN performs the reconstruct. However, `xdms`(8) can resume monitoring status of the reconstruct by reissuing the reconstruct or issuing an `info` request on the array device. The `info` request will indicate the percent of reconstruct complete if there is a reconstruct currently in process.
- The addresses of array members cannot be changed. If the array was initialized as units 1 through 5, it cannot be moved to 9 through 13 (011 through 015), 17 through 21 (021 through 025), etc. It can be moved to another channel as long as the target addresses are the same. When you want to change the addresses, you must execute `xdms -a init` and remake the `/dev/xdd` nodes.

Remote Mount [10]

The UNICOS/mk operating system supports the *remote mount* feature. This feature allows you to configure your system with multiple file servers, which in turn allows you to mount file systems remotely on OS PEs other than the OS PE that contains the root file server. This method of configuration provides the following advantages:

- A file server can be located on the same OS PE as any disk server.
- Multiple file servers can handle more requests than a single file server.

10.1 Remote Mount Configuration

To configure a remote file server, make the following entries in the configuration file:

- Define an OS PE which contains a file actor and, in most cases, a disk actor.
- Configure the attached disk to use the disk actor.
- Configure the `ldd` device (by means of the `file_pe` field) to use the file server.

When configuring a system with multiple file servers, the following considerations hold:

- A system must have one root file server, which services at least the root file system.
- If the `dsk[]` entry for the root device has the `file_pe` field defined, you must also define and match this with the `file_pe` field in the `rootdev` structure. For example:

```
dev {
     ...
     dsk[3] {
            minor = 77; name = "root"; nslices = 1;
            member[0] {
                    mtype = m_xdd;
                    mstor { phys = &mf[0].dev.xdd[2]; }
            }
            file_pe = "ospe_b"; /* must match that of rootdev */
```

```
      }
}

rootdev { name = "root"; minor = 77; file_pe = "ospe_b"; }
```

- It is not necessary that a file server be located on the same OS PE as a disk server, but locating the file and disk servers on the same OS PE reduces messaging.
- Any `ldd` device that does not have its `file_pe` set uses the root file server.
- Both block device access and mounted file system access use the remote file server.

10.2 Remote Mount Limitations

The remote mount feature has the following limitations:

- You cannot mount a file system on a remotely mounted file system.
- There is no NFS export of a remotely mounted file system.

When a system call has a *pathname* argument, it needs to look up the name to resolve the argument to a specific file. When this *lookup* needs to go to a new file system (across a mount point), it is called a *traversal*. Each time a traversal goes from one file server to another, it is a remote hop.

For example, consider the following file system setup:

`/usr` is a file system mounted on the root file server.

`/usr/src` is a file system mounted on a remote file server.

`/usr/abc` is a directory on `/usr`.

`/usr/src/bsd` is a directory on `/usr/src`.

With the above setup:

A lookup of `/usr/src/bsd/file1` has one remote hop.

A lookup of `/usr/abc/file2` has no remote hops.

A lookup of `/usr/abc/../src/../abc/../src/bsd/file1` has 3 remote hops.

- The maximum number of symbolic links in a lookup is limited to `v_symlinkmax`. The default maximum is 20.

- The maximum number of remote hops in a lookup is limited to `v_remotehops`. The default maximum is 20.

10.3 Configuration Example Using Remote Mount

This section provides an example of a configuration file for a system that incorporates multiple packet, disk and file servers in a UNICOS/mk system.

With the support of node-based disk configuration in the UNICOS/mk system, configuration of multiple disk servers was simplified. It is necessary to provide `pdisk` definitions for the root device only. Multiple disk driver structures can be used to define multiple disk servers instead of many `pdisk` entries. You can still use `pdisk` entries, however, if you want to assign disk servers to individual disks.

When configuring a system to use the remote mount feature, the file system name, the minor device number, and the `file_pe` are needed. The other information about the device in the configuration file is not used; the system gets this information from the node.

The following sections provide examples of two different ways to configure multiple packet, disk, and file servers in a node-based system.

10.3.1 Example 1

```
/* USMID @(#)srv/nmakefiles/config/sn6549 30.33 09/25/97 09:17:33 */

/*
 * (C) COPYRIGHT CRAY RESEARCH, INC.
 * UNPUBLISHED PROPRIETARY INFORMATION.
 * ALL RIGHTS RESERVED.
 */

mainframe_t mf[1];

mf[0] {
 os {
  nbuf = 5000;
  sys = "sn6549";
  node = "acidrain";
  ipc {
   msg {
```

```
  msgmax = 2048;
  msgmnb = 4096;
  msgmni = 136;
  msgssz = 32;
  msgtql = 544;
  msgseg = 4352;
 }
 sem {
  semmni = 272;
  semmns = 544;
  semmnu = 272;
  semmsl = 25;
  semopm = 25;
  semume = 20;
  semvmx = 32767;
  semaem = 16384;
 }
 shm {
  shmmax = 524288;
  shmmin = 1;
  shmmni = 200;
  shmseg = 10;
 }
}
nc1_min_raw = 20;
panicflush = 1;
extdcore = 0;
shlbmax = 230;
rlimits {
 rlim_use = 0;
 rlim_cur = 0;
 rlim_max = 0;
 rlim_cpu_use = 0;
 rlim_cpu_cur = 2147483647;
 rlim_cpu_max = 2147483647;
 rlim_mem_use = 0;
 rlim_mem_cur = 4611686018427387903;
 rlim_mem_max = 4611686018427387903;
 rlim_nop_use = 0;
 rlim_nop_cur = 10000;
 rlim_nop_max = 10000;
 rlim_sds_use = 0;
 rlim_sds_cur = 4611686018427387903;
```

```
rlim_sds_max = 4611686018427387903;
rlim_cproc_use = 0;
rlim_cproc_cur = 100;
rlim_cproc_max = 1000;
rlim_cor_use = 0;
rlim_cor_cur = 1048576;
rlim_cor_max = 1048576;
rlim_tape0_use = 0;
rlim_tape0_cur = 255;
rlim_tape0_max = 255;
rlim_tape1_use = 0;
rlim_tape1_cur = 255;
rlim_tape1_max = 255;
rlim_tape2_use = 0;
rlim_tape2_cur = 255;
rlim_tape2_max = 255;
rlim_tape3_use = 0;
rlim_tape3_cur = 255;
rlim_tape3_max = 255;
rlim_tape4_use = 0;
rlim_tape4_cur = 255;
rlim_tape4_max = 255;
rlim_tape5_use = 0;
rlim_tape5_cur = 255;
rlim_tape5_max = 255;
rlim_tape6_use = 0;
rlim_tape6_cur = 255;
rlim_tape6_max = 255;
rlim_tape7_use = 0;
rlim_tape7_cur = 255;
rlim_tape7_max = 255;
rlim_mppe_use = 0;
rlim_mppe_cur = 256;
rlim_mppe_max = 1024;
rlim_sz_use = 0;
rlim_sz_cur = 2147483647;
rlim_sz_max = 2147483647;
rlim_mppt_use = 0;
rlim_mppt_cur = 4611686018427387903;
rlim_mppt_max = 4611686018427387903;
rlim_mppb_use = 0;
rlim_mppb_cur = 1;
rlim_mppb_max = 63;
```

```
      rlim_rsblk_use = 0;
      rlim_rsblk_cur = 64;
      rlim_rsblk_max = 1024;
      rlim_shm_num_use = 0;
      rlim_shm_num_cur = 1;
      rlim_shm_num_max = 10;
      rlim_shm_size_use = 0;
      rlim_shm_size_cur = 200;
      rlim_shm_size_max = 524288;
      rlim_fdm_use = 0;
      rlim_fdm_cur = 64;
      rlim_fdm_max = 1024;
      rlim_sockbf_use = 0;
      rlim_sockbf_cur = 450;
      rlim_sockbf_max = 450;
      rlim_data_use = 0;
      rlim_data_cur = 536870912;
      rlim_data_max = 536870912;
      rlim_stack_use = 0;
      rlim_stack_cur = 536870912;
      rlim_stack_max = 536870912;
      rlim_vmem_use = 0;
      rlim_vmem_cur = 536870912;
      rlim_vmem_max = 536870912;
      rlim_mppm_use = 0;
      rlim_mppm_cur = 4611686018427387903;
      rlim_mppm_max = 4611686018427387903;
     }
     mclock {
      clock_major = 30;
      proclim_extra_time = 30;
      job_cmd_major = 300;
      job_mpp_major = 30;
      joblim_extra_time = 31;
      incacct_major = 600;
     }
    }
    nfs_info {
     nfs_num_rnodes = 256;
     nfs_maxdata = 32768;
     nfs_static_clients = 8;
     nfs_temp_clients = 8;
     cnfs_static_clients = 8;
```

```
 cnfs_temp_clients = 8;
 nfs_maxdupreqs = 1200;
 nfs_duptimeout = 3;
 nfs_printinter = 0;
 nfs_wcredmax = 0;
}
network {
 nw_dev[0] {
  iopath { ioc = 0; iop = 1; chan = 1; }
  ordinal = 0; type = nw_fddi;

  /* kernel defaults
  maxusers = 1; maxoutputs = 16;
  maxinputs = 16; mtu = 4352;
  */
 }
 tcp_nmbspace = 10000;
}

/*
        * Logical devices - file system configuration. Remote mount
        * (multiple file server) configuration is done here.  The
        * name, minor, nslices, and mtype entries are required for
        * remote mount.  The "mstor {phys =  ...}" entries are optional
        * for remote mount.  File systems not specifically assigned to a
        * file server default to the file server of the root file system.
        */

dev {
 dsk[0] {
  minor = 230; name = "root.usr";
  nslices = 1; file_pe = "ospe_c";
  member[0] {
   mtype = m_xdd;
   mstor { phys = &mf[0].dev.xdd[0]; }
  }
 }
 dsk[1] {
  minor = 233; name = "root.usr_bu";
  nslices = 1; file_pe = "ospe_c";
  member[0] {
   mtype = m_xdd;
   mstor { phys = &mf[0].dev.xdd[1]; }
```

```
 }
}
dsk[2] {
 minor = 155; name = "dd308.root"; nslices = 1;
 file_pe = "ospe_d";
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf[0].dev.xdd[2]; }
 }
}
dsk[3] {
 minor = 160; name = "dd308.root.bu"; nslices = 1;
 file_pe = "ospe_d";
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf[0].dev.xdd[3]; }
 }
}

/*
  * The name, minor device number, and file_pe are needed
 * by remote mount.
 */

dsk[4:12] { file_pe = "ospe_c"; }

dsk[4]  { minor = 20;  name = "opt"; }  /* /opt */
dsk[5]  { minor = 21;  name = "tmp"; }  /* /usr/tmp */
dsk[6]  { minor = 241; name = "usr_adm"; } /* /usr/adm */
dsk[7]  { minor = 242; name = "usr_spool"; } /* /usr/spool */
dsk[8]  { minor = 6;   name = "admin"; } /* admin */
dsk[10] { minor = 244; name = "qte1"; }  /* /qte1 */
dsk[11] { minor = 215; name = "ptmp"; }  /* /ptmp */
dsk[12] { minor = 9;   name = "home"; }  /* /acidrain */

/* MPN 1 slices */

xdd[0] {
 minor = 230; name = "root.usr";
 start = 25000; length = 500000; length_unit = blocks;
 pstor = &mf[0].disk_info.phys_stor[0];
}
xdd[1] {
```

```
  minor = 233; name = "root.usr_bu";
  start = 25000; length = 500000; length_unit = blocks;
  pstor = &mf[0].disk_info.phys_stor[1];
 }

 /* FCN slices */

 xdd[2] {
  minor = 155; name = "dd308.root";
  start = 25000; length = 200000; length_unit = blocks;
  pstor = &mf[0].disk_info.phys_stor[2];
 }
 xdd[3] {
  minor = 160; name = "dd308.root.bu";
  start = 25000; length = 200000; length_unit = blocks;
  pstor = &mf;[0].disk_info.phys_stor[3];
 }
}

/*
        * All I/O nodes not specifically assigned to a disk driver (server)
        * will default to the disk driver opening the root file system.
        * Assign the node number of the I/O node (not the mainframe node).
        * These assignments can be superceded by the phys_stor definitions.
        */

drivers {
 diskdriver[0] {
  pchdrs = 2000; /* # cache headers (units) */
  pccore = 12000; /* number of blocks for pcache */
  pcsync = 900; /* 15 minute sync interval */
  pcmaxdevs = 100;/* max # of devices cachable */
  pddmax = 32;
  xddmax = 256;
  pddslmax = 256;
  mddslmax = 8;
  sddslmax = 8;
  rddslmax = 4;
  xddslmax = 256;
  ion[0] { ring = 0; node = 1; }
  ion[1] { ring = 1; node = 1; }
  ion[2] { ring = 2; node = 1; }
  ion[3] { ring = 3; node = 1; }
```

```
 disk_pe = "ospe_c";
 }
 diskdriver[1] {
  pchdrs = 2000;
  pccore = 12000;
  pcsync = 900;
  pcmaxdevs = 100;
  pddmax = 32;
  xddmax = 256;
  pddslmax = 256;
  mddslmax = 8;
  sddslmax = 8;
  rddslmax = 4;
  xddslmax = 256;
  ion[0] { ring = 4; node = 1; }
  disk_pe = "ospe_d";
 }
 tapedriver {
  tpd_max_bufz = 262144;
  tpd_max_conf_up = 8;
  tpd_maxdev = 8;
 }
}
disk_info {
 /* MPN 0 disks */

 phys_stor[0] {
  name = "01011.01_dd314"; disk_pe = "ospe_c";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 1; iop = 1; chan = 1; }
  unit = 1;
 }
 phys_stor[1] {
  name = "01013.01_dd314"; disk_pe = "ospe_c";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 1; iop = 1; chan = 3; }
  unit = 1;
 }

 /* FCN disks */

 phys_stor[2] {
  name = "dd.04010.05"; disk_pe = "ospe_d";
```

```
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 4; iop = 1; chan = 0; }
  unit = 5;
 }
 phys_stor[3] {
  name = "dd.04014.044"; disk_pe = "ospe_d";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 4; iop = 1; chan = 4; }
  unit = 36;
 }
}
rootdev { name = "dd308.root"; minor = 155;
   file_pe = "ospe_d"; }
swapdev { name = "dd308.swap.str"; minor = 153; }
boot_info {
 rev_level = 1;
 boot_info_size = 264;
 mf_name = "NULL";
 scx_ring = 0;
 scx_node = 15;
 scx_chan = 0;
 scx_dest_node = 1;
 archive_size = 0;
 mkpal_size = 0;
 config_size = 0;
 clock {
  month = 0;
  day = 0;
  year = 0;
  hour = 0;
  min = 0;
  sec = 0;
  tz = 0;
 }
 num_pes = 68;
 run_level = 0;
 boot_pe = 0x043;
 boot_options = 0x00;
 sn = 6549;
 hz = 75000000;
}

/*
```

```
 * Gigaring configuration
 *
 * Assign the node number of the mainframe node (not the
 * I/O node) for the I/O control ppe (lpe).
 */

/* GigaRing info - mpn0 - slot 1 */

routes[0] {
 ringno = 0;
 io_contr_ppe = 0x040000;
 gigaring_node_addr = 15;
 path_type = primary;
 link_type = link_scx;
 intr_ps_name = "ospe_b";
 send_ps_name = "ospe_b";
}

/* GigaRing info - mpn1 - 10002, 10003, 10102, 10103 */

routes[1] {
 ringno = 1;
 io_contr_ppe = 0x010002;
 gigaring_node_addr = 2;
 path_type = primary;
 link_type = link_scx;
 intr_ps_name = "ospe_c";
 send_ps_name = "ospe_c";
}

/* GigaRing info - mpn2 - 20202, 20203, 20302, 20303 */

routes[2] {
 ringno = 2;
 io_contr_ppe = 0x020202;
 gigaring_node_addr = 2;
 path_type = primary;
 link_type = link_scx;
 intr_ps_name = "ospe_c";
 send_ps_name = "ospe_c";
}

/* GigaRing info - mpn3 - 200, 201, 300, 301 */
```

```
routes[3] {
 ringno = 3;
 io_contr_ppe = 0x000200;
 gigaring_node_addr = 2;
 path_type = primary;
 link_type = link_scx;
 intr_ps_name = "ospe_c";
 send_ps_name = "ospe_c";
}

/* GigaRing info - fcn0 - 000, 001, 100, 101 */

routes[4] {
 ringno = 4;
 io_contr_ppe = 0x000000;
 gigaring_node_addr = 2;
 path_type = primary;
 link_type = link_scx;
 intr_ps_name = "ospe_d";
 send_ps_name = "ospe_d";
}

/*
        * Server Routing
        *
        * If the disk server and packet server are on the same PE, a
        * server_routes[] specification is not needed.
        */

server_routes[0] {
 ringno = 0;
 server_name = "disk";
 server_pe_name = "ospe_c";
 ps_pe_name = "ospe_b";
}
mpp {
 pe[0:47] {
  flags = 0;
  pe_type = PE_TYPE_APP;
  actor_list_index = 1;
 }
 pe[48:63] {
```

```
 flags = 0;
 pe_type = PE_TYPE_CMD;
 actor_list_index = 0;
}
pe[64:67] {
 flags = 0;
 pe_type = PE_TYPE_OS;
}
pe[64] { /* disk, swap i/o & root file server */
 actor_list_index = 8;
 pe_name = "ospe_d";
}
pe[65] { /* disk, tape, & file i/o */
 actor_list_index = 7;
 pe_name = "ospe_c";
}
pe[66] { /* networking i/o */
 actor_list_index = 6;
 pe_name = "ospe_b";
}
pe[67] { /* boot & general */
 pe_index = 0x043; /* temporary */
 actor_list_index = 5;
 pe_name = "ospe_a";
}

/* Command PEs */

pe_actors[0] {
 actor_name[0] = "kernel";
 actor_name[1] = "PM";
 actor_name[2] = "fsa";
 actor_name[3] = "em";
}

/* Application PEs */

pe_actors[1] {
 actor_name[0] = "kernel";
 actor_name[1] = "PM";
 actor_name[2] = "fsa";
 actor_name[3] = "em";
}
```

```
/* OS PE - for single-OS-PE systems */

pe_actors[2] {
 actor_name[0] = "kernel";
 actor_name[1] = "PM";
 actor_name[2] = "fsa";
 actor_name[3] = "log";
 actor_name[4] = "info";
 actor_name[5] = "em";
 actor_name[6] = "config";
 actor_name[7] = "packet";
 actor_name[8] = "disk";
 actor_name[9] = "tty";
 actor_name[10] = "netdev";
 actor_name[11] = "file";
 actor_name[12] = "socket";
 actor_name[13] = "alm";
 actor_name[14] = "grm";
 actor_name[15] = "GPM";
 actor_name[16] = "ipcm";
 actor_name[17] = "UP_INIT";
 actor_name[18] = "sio";
 actor_name[19] = "tape";
}

/* OS PE - two-OS-PE system - first (boot) OS PE */

pe_actors[3] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "config";
 actor_name[4] = "info";
 actor_name[5] = "log";
 actor_name[6] = "grm";
 actor_name[7] = "alm";
 actor_name[8] = "tty";
 actor_name[9] = "GPM";
 actor_name[10] = "ipcm";
 actor_name[11] = "UP_INIT";
}
```

```
/* OS PE - two-OS-PE system - second OS PE */

pe_actors[4] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "packet";
 actor_name[4] = "disk";
 actor_name[5] = "netdev";
 actor_name[6] = "tape";
 actor_name[7] = "file";
 actor_name[8] = "socket";
 actor_name[9] = "sio";
}

/* OS PE - four-OS-PE system - boot & general OS PE */

pe_actors[5] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "config";
 actor_name[4] = "info";
 actor_name[5] = "log";
 actor_name[6] = "grm";
 actor_name[7] = "alm";
 actor_name[8] = "GPM";
 actor_name[9] = "ipcm";
 actor_name[10] = "UP_INIT";
}

/* OS PE - four-OS-PE system - networking OS PE */

pe_actors[6] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "packet";
 actor_name[4] = "netdev";
 actor_name[5] = "socket";
 actor_name[6] = "tty";
}
```

```
/* OS PE - four-OS-PE system - disk, tape, & file OS PE*/

pe_actors[7] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "packet";
 actor_name[4] = "disk";
 actor_name[5] = "tape";
 actor_name[6] = "file";
}

/* OS PE - four-OS-PE system - disk, file, & swap OS PE */

pe_actors[8] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "packet";
 actor_name[4] = "disk";
 actor_name[5] = "file";
 actor_name[6] = "sio";
}

/* OS PE - Additional Disk Server PE */

pe_actors[9] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "packet";
 actor_name[4] = "disk";
}

/* OS PE - Additional Network Server PE */

pe_actors[10] {
 actor_name[0] = "kernel";
 actor_name[1] = "em";
 actor_name[2] = "PM";
 actor_name[3] = "packet";
 actor_name[4] = "netdev";
 actor_name[5] = "socket";
```

```
 }
}
audit_info {
 state = 0;
 all_nami = 0;
 all_rm = 0;
 all_valid = 0;
 audit = 1;
 chdir = 1;
 config = 1;
 crl = 0;
 dac = 1;
 discv = 1;
 filexfr = 1;
 physio_err = 0;
 ipnet = 1;
 jend = 1;
 jstart = 1;
 linkv = 1;
 mandv = 1;
 mkdirv = 1;
 netcf = 0;
 netwv = 1;
 nfs = 0;
 nqs = 0;
 nqscf = 0;
 path_track = 1;
 oper = 1;
 priv = 0;
 removev = 1;
 rmdirv = 1;
 secsys = 1;
 suid = 1;
 shutdown = 1;
 startup = 1;
 sulog = 1;
 tape = 0;
 user = 0;
 trust = 0;
 tchg = 1;
 panic = 1;
 maxsize = 8192000;
 dir = "/usr/adm/sl";
```

```
 file = "slogfile";
 fprefix = "s.";
}
var {
 nbuf = 5000;
 nbuf_fctr = 20;
 nhbuf_fctr = 4;
 nblk_fctr = 20;
 npbuf = 500;
 nasyn = 800;
 maxasyn = 192;
 nsu_asyn = 4;
 maxpipe = 20;
 maxrah = 8;
 ncrbuf = 4;
 link_max = 1000;
 nldmap = 250;
 nmnt = 250;
 nmt = 0;
 nldch = 0;
 ldchcore = 0;
 nsidebuf = 48;
 npty = 128;
 nc1minraw = 20;
 nusers = 200;
 njobs = 300;
 nproc = 650;
 ncall = 650;
 nc_namlen = 15;
 nexecs = 4;
 ntext = 100;
 nclist = 1000;
 acnice = 1;
 cdlimit = 040000000;
 max_unlinked_bytes = 26214400;
 dmode = 1;
 maxup = 100;
 nmount = 75;
 nfile = 3000;
 ninode = 1500;
 nc1inode = 1500;
 nc_size = 1024;
 nquota = 0;
```

```
 nflocks = 100;
 user_mls = 1;
}
security {
 secure_scrub = 0;
 priv_su = 1;
 nfs_secure_export_ok = 1;
 nfs_remote_rw_ok = 1;
 minslevel = 0;
 dev_enforce_on = 0;
 secure_mac = 0;
 secure_net_options = 6;
 forced_socket = 0;
 random_pass_on = 0;
 pass_minsize = 8;
 pass_maxsize = 8;
 maxlogs = 5;
 logdelay = 0;
 delay_mult = 0;
 disable_time = 0;
 console_msg = 0;
 disable_acct = 0;
 maxslevel = 0;
 sysvcomps = 0;
 secure_system_console = "";
 system_admin_console = "/dev/console";
 secure_operator_console = "/dev/console";
 declassify_disk = 0;
 overwrite_count = 3;
 declassify_pattern = 0;
 sanitize_pattern = 0;
 secure_pipe = 0;
 fsetid_restrict = 0;
 mls_obj_ranges = 0;
 comparts[0] = "secadm";
 comparts[1] = "sysadm";
 comparts[2] = "sysops";
 comparts[3] = "netadm";
 comparts[4] = "unicos";
 comparts[5] = "crayri";
 comparts[6] = "comp24";
 comparts[7] = "comp39";
 comparts[8] = "comp63";
```

```
compbits[0] = 01;
compbits[1] = 02;
compbits[2] = 04;
compbits[3] = 010;
compbits[4] = 020;
compbits[5] = 040;
compbits[6] = 040000000;
compbits[7] = 04000000000000;
compbits[8] = 0400000000000000000000;
lvlname[0] = "level0";
lvlname[1] = "level1";
lvlname[2] = "level2";
lvlname[3] = "level3";
lvlname[4] = "level4";
lvlname[5] = "level5";
lvlname[6] = "level6";
lvlname[7] = "level7";
lvlname[8] = "level8";
lvlname[9] = "level9";
lvlname[10] = "level10";
lvlname[11] = "level11";
lvlname[12] = "level12";
lvlname[13] = "level10";
lvlname[14] = "level14";
lvlname[15] = "level15";
lvlname[16] = "level16";
lvlnums[0] = 0;
lvlnums[1] = 1;
lvlnums[2] = 2;
lvlnums[3] = 3;
lvlnums[4] = 4;
lvlnums[5] = 5;
lvlnums[6] = 6;
lvlnums[7] = 7;
lvlnums[8] = 8;
lvlnums[9] = 9;
lvlnums[10] = 10;
lvlnums[11] = 11;
lvlnums[12] = 12;
lvlnums[13] = 13;
lvlnums[14] = 14;
lvlnums[15] = 15;
lvlnums[16] = 16;
```

```
 }
}

END_CONFIG
```

10.3.2 Example 2

```
/* USMID @(#)srv/nmakefiles/config/sn6501 30.43 09/25/97 09:17:33 */

/*
 * (C) COPYRIGHT CRAY RESEARCH, INC.
 * UNPUBLISHED PROPRIETARY INFORMATION.
 * ALL RIGHTS RESERVED.
 */

/*
 * Skeleton UNICOS/MK parameter file - sn6501
 *
 * Notes:
 * - there are 8 disks configured, all DD314s
 * - DD314s are 1,102,256 blocks
 * - Colin Ngam has stated block 1,101,999 should be the last allocated
 * - blocks 1,102,000 to 1,102,255 are for C.E. usage
 */

mainframe_t mf[1];

mf[0] {
 os {
  nbuf = 2000;
  sys = "sn6501";
  node = "galileo";
  ipc {
   msg {
    msgmax = 2048;
    msgmnb = 4096;
    msgmni = 16;
    msgssz = 32;
    msgtql = 64;
    msgseg = 512;
   }
   sem {
```

```
  semmni = 32;
  semmns = 64;
  semmnu = 32;
  semmsl = 25;
  semopm = 25;
  semume = 20;
  semvmx = 32767;
  semaem = 16384;
 }
 shm {
  shmmax = 524288;
  shmmin = 1;
  shmmni = 200;
  shmseg = 10;
 }
}
nc1_min_raw = 20;
panicflush = 0;
extdcore = 0;
shlbmax = 230;
rlimits {
 rlim_use = 0;
 rlim_cur = 0;
 rlim_max = 0;
 rlim_cpu_use = 0;
 rlim_cpu_cur = 2147483647;
 rlim_cpu_max = 2147483647;
 rlim_mem_use = 0;
 rlim_mem_cur = 4611686018427387903;
 rlim_mem_max = 4611686018427387903;
 rlim_nop_use = 0;
 rlim_nop_cur = 10000;
 rlim_nop_max = 10000;
 rlim_sds_use = 0;
 rlim_sds_cur = 4611686018427387903;
 rlim_sds_max = 4611686018427387903;
 rlim_cproc_use = 0;
 rlim_cproc_cur = 100;
 rlim_cproc_max = 1000;
 rlim_cor_use = 0;
 rlim_cor_cur = 1048576;
 rlim_cor_max = 1048576;
 rlim_tape0_use = 0;
```

```
rlim_tape0_cur = 255;
rlim_tape0_max = 255;
rlim_tape1_use = 0;
rlim_tape1_cur = 255;
rlim_tape1_max = 255;
rlim_tape2_use = 0;
rlim_tape2_cur = 255;
rlim_tape2_max = 255;
rlim_tape3_use = 0;
rlim_tape3_cur = 255;
rlim_tape3_max = 255;
rlim_tape4_use = 0;
rlim_tape4_cur = 255;
rlim_tape4_max = 255;
rlim_tape5_use = 0;
rlim_tape5_cur = 255;
rlim_tape5_max = 255;
rlim_tape6_use = 0;
rlim_tape6_cur = 255;
rlim_tape6_max = 255;
rlim_tape7_use = 0;
rlim_tape7_cur = 255;
rlim_tape7_max = 255;
rlim_mppe_use = 0;
rlim_mppe_cur = 256;
rlim_mppe_max = 1024;
rlim_sz_use = 0;
rlim_sz_cur = 2147483647;
rlim_sz_max = 2147483647;
rlim_mppt_use = 0;
rlim_mppt_cur = 4611686018427387903;
rlim_mppt_max = 4611686018427387903;
rlim_mppb_use = 0;
rlim_mppb_cur = 1;
rlim_mppb_max = 63;
rlim_rsblk_use = 0;
rlim_rsblk_cur = 64;
rlim_rsblk_max = 1024;
rlim_shm_num_use = 0;
rlim_shm_num_cur = 1;
rlim_shm_num_max = 10;
rlim_shm_size_use = 0;
rlim_shm_size_cur = 200;
```

```
  rlim_shm_size_max = 524288;
  rlim_fdm_use = 0;
  rlim_fdm_cur = 64;
  rlim_fdm_max = 1024;
  rlim_sockbf_use = 0;
  rlim_sockbf_cur = 450;
  rlim_sockbf_max = 450;
  rlim_data_use = 0;
  rlim_data_cur = 1073741824;
  rlim_data_max = 1073741824;
  rlim_stack_use = 0;
  rlim_stack_cur = 1073741824;
  rlim_stack_max = 1073741824;
  rlim_vmem_use = 0;
  rlim_vmem_cur = 1073741824;
  rlim_vmem_max = 1073741824;
  rlim_mppm_use = 0;
  rlim_mppm_cur = 4611686018427387903;
  rlim_mppm_max = 4611686018427387903;
 }
 mclock {
  clock_major = 30;
  proclim_extra_time = 30;
  job_cmd_major = 300;
  job_mpp_major = 30;
  joblim_extra_time = 31;
  incacct_major = 600;
 }
}
nfs_info {
 nfs_num_rnodes = 256;
 nfs_maxdata = 32768;
 nfs_static_clients = 8;
 nfs_temp_clients = 8;
 cnfs_static_clients = 8;
 cnfs_temp_clients = 8;
 nfs_maxdupreqs = 1200;
 nfs_duptimeout = 3;
 nfs_printinter = 0;
 nfs_wcredmax = 0;
}
network {
 nw_dev[0] {
```

```
    iopath { ioc = 0; iop = 1; chan = 1; }
    ordinal = 0; type = nw_fddi;

    /*
     * The following lines are commented out to
     * allow for the kernel default to be used.
     */

    /*
     maxusers = 1; maxoutputs = 16;
     maxinputs = 16; othreshold = 0;
     ithreshold = 0; mtu = 4352;
     */
   }
   nw_dev[1] {
    iopath { ioc = 0; iop = 1; chan = 6; }
    ordinal = 0; type = nw_ethernet;

    /*
     * The following lines are commented out to
     * allow for the kernel default to be used.
     */

    /*
     maxusers = 1; maxoutputs = 16;
     maxinputs = 16; othreshold = 0;
     ithreshold = 0; mtu = 1500;
     */
   }
   tcp_nmbspace = 4000;
  }
  dev {
   dsk[0] {
    minor = 1; name = "scratch.s40"; nslices = 1;
    member[0] {
     mtype = m_xdd;
     mstor { phys = &mf[0].dev.xdd[0]; }
    }
   }
   dsk[1] {
    minor = 2; name = "root"; nslices = 1;
    member[0] {
     mtype = m_xdd;
```

```
  mstor { phys = &mf;[0].dev.xdd[1]; }
 }
}
dsk[2] {
 minor = 3; name = "usr_bu"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[2]; }
 }
}
dsk[3] {
 minor = 4; name = "usr_adm"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[3]; }
 }
}
dsk[4] {
 minor = 5; name = "usr_spool"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[4]; }
 }
}
dsk[5] {
 minor = 6; name = "admin"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[5]; }
 }
}
dsk[6] {
 minor = 7; name = "dumps"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[6]; }
 }
}
dsk[7] {
 minor = 8; name = "scratch.s50"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[7]; }
```

```
 }
}
dsk[8] {
 minor = 9; name = "root_bu"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[8]; }
 }
}
dsk[9] {
 minor = 10; name = "usr"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[9]; }
 }
}
dsk[10] {
 minor = 11; name = "open.s50a"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[10]; }
 }
}
dsk[11] {
 minor = 12; name = "dump"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[11]; }
 }
}
dsk[12] {
 minor = 13; name = "qtest1"; nslices = 3;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[12]; }
 }
 member[1] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[13]; }
 }
 member[2] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[14]; }
```

```
 }
}
dsk[13] {
 minor = 16; name = "qtest2"; nslices = 2;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[15]; }
 }
 member[1] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[16]; }
 }
}
dsk[14] {
 minor = 18; name = "qtest3"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[17]; }
 }
}
dsk[15] {
 minor = 19; name = "qte1"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[18]; }
 }
}
dsk[16] {
 minor = 20; name = "opt"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[19]; }
 }
}
dsk[17] {
 minor = 21; name = "scratch.s60"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[20]; }
 }
}
dsk[18] {
 minor = 22; name = "swap"; nslices = 1;
```

```
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[21]; }
 }
}
dsk[19] {
 minor = 23; name = "home"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[22]; }
 }
}
dsk[20] {
 minor = 24; name = "scratch.s30"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[23]; }
 }
}
dsk[21] {
 minor = 25; name = "ptmp"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[24]; }
 }
}
dsk[22] {
 minor = 26; name = "scratch.s41"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[25]; }
 }
}
dsk[23] {
 minor = 27; name = "tmp.s41"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[26]; }
 }
}
dsk[24] {
 minor = 28; name = "scratch.s51"; nslices = 1;
 member[0] {
```

```
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[27]; }
 }
}
dsk[25] {
 minor = 29; name = "tmp.s51"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[28]; }
 }
}
dsk[26] {
 minor = 30; name = "scratch.s61"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[29]; }
 }
}
dsk[27] {
 minor = 31; name = "tmp.s61"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[30]; }
 }
}
dsk[28] {
 minor = 32; name = "scratch.s31"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[31]; }
 }
}
dsk[29] {
 minor = 33; name = "tmp.s31"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[32]; }
 }
}
dsk[30] {
 minor = 101; name = "ramroot"; nslices = 1;
 member[0] {
  mtype = m_rdd;
```

```
  mstor { phys = &mf;[0].dev.rdd[0]; }
 }
}
dsk[31] {
 minor = 102; name = "ramswap"; nslices = 1;
 member[0] {
  mtype = m_rdd;
  mstor { phys = &mf;[0].dev.rdd[1]; }
 }
}
dsk[32] {
 minor = 35; name = "target.root"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[34]; }
 }
}
dsk[33] {
 minor = 34; name = "src.s31"; nslices = 1;
 member[0] {
  mtype = m_xdd;
  mstor { phys = &mf;[0].dev.xdd[33]; }
 }
}
rdd[0] {
 minor = 1; name = "ramroot";
 start = 0; length = 8192; length_unit = blocks;
 pstor = &mf;[0].disk_info.ram_stor[0];
}
rdd[1] {
 minor = 2; name = "ramswap";
 start = 8192; length = 1; length_unit = blocks;
 pstor = &mf;[0].disk_info.ram_stor[0];
}
xdd[0] {
 minor = 1; name = "scratch.s40";
 start = 0; length = 12000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
}
xdd[1] {
 minor = 2; name = "root";
 start = 12000; length = 200000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
```

```
}
xdd[2] {
 minor = 3; name = "usr_bu";
 start = 212000; length = 250000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
}
xdd[3] {
 minor = 4; name = "usr_adm";
 start = 462000; length = 75000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
}
xdd[4] {
 minor = 5; name = "usr_spool";
 start = 537000; length = 75000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
}
xdd[5] {
 minor = 6; name = "admin";
 start = 612000; length = 50000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
}
xdd[6] {
 minor = 7; name = "dumps";
 start = 662000; length = 439999; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[0];
}
xdd[7] {
 minor = 8; name = "scratch.s50";
 start = 0; length = 12000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[8] {
 minor = 9; name = "root_bu";
 start = 12000; length = 200000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[9] {
 minor = 10; name = "usr";
 start = 212000; length = 250000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[10] {
 minor = 11; name = "open.s50a";
```

```
 start = 462000; length = 50000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[11] {
 minor = 12; name = "dump";
 start = 514000; length = 35000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[12] {
 minor = 13; name = "qtest1_1";
 start = 549000; length = 7000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[13] {
 minor = 14; name = "qtest1_2";
 start = 556000; length = 7000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[14] {
 minor = 15; name = "qtest1_3";
 start = 563000; length = 7000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[15] {
 minor = 16; name = "qtest2_1";
 start = 570000; length = 7000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[16] {
 minor = 17; name = "qtest2_2";
 start = 577000; length = 7000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[17] {
 minor = 18; name = "qtest3";
 start = 584000; length = 7000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[18] {
 minor = 19; name = "qte1";
 start = 591000; length = 250000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
```

```
xdd[19] {
 minor = 20; name = "opt";
 start = 841000; length = 260999; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[2];
}
xdd[20] {
 minor = 21; name = "scratch.s20";
 start = 0; length = 12000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[4];
}
xdd[21] {
 minor = 22; name = "swap";
 start = 12000; length = 525000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[4];
}
xdd[22] {
 minor = 23; name = "home";
 start = 537000; length = 564999; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[4];
}
xdd[23] {
 minor = 24; name = "scratch.s30";
 start = 0; length = 12000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[6];
}
xdd[24] {
 minor = 25; name = "ptmp";
 start = 12000; length = 1089999; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[6];
}
xdd[25] {
 minor = 26; name = "scratch.s41";
 start = 0; length = 12000; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[1];
}
xdd[26] {
 minor = 27; name = "tmp.s41";
 start = 12000; length = 1089999; length_unit = blocks;
 pstor = &mf;[0].disk_info.phys_stor[1];
}
xdd[27] {
 minor = 28; name = "scratch.s51";
 start = 0; length = 12000; length_unit = blocks;
```

```
     pstor = &mf;[0].disk_info.phys_stor[3];
    }
    xdd[28] {
     minor = 29; name = "tmp.s51";
     start = 12000; length = 726666; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[3];
    }
    xdd[29] {
     minor = 30; name = "scratch.s21";
     start = 0; length = 12000; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[5];
    }
    xdd[30] {
     minor = 31; name = "tmp.s21";
     start = 12000; length = 1089999; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[5];
    }
    xdd[31] {
     minor = 32; name = "scratch.s31";
     start = 0; length = 12000; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[7];
    }
    xdd[32] {
     minor = 33; name = "tmp.s31";
     start = 750666; length = 363333; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[7];
    }
    xdd[33] {
     minor = 34; name = "src.s31";
     start = 12000; length = 738666; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[7];
    }
    xdd[34] {
     minor = 35; name = "target.root";
     start = 738666; length = 363333; length_unit = blocks;
     pstor = &mf;[0].disk_info.phys_stor[3];
    }
   }
   drivers {
    diskdriver[0] {
     pchdrs = 512;
     pccore = 4096;
     pcsync = 300;
```

```
  pcmaxdevs = 256;
  pddmax = 32;
  xddmax = 256;
  pddslmax = 256;
  rddslmax = 4;
  xddslmax = 256;
 }
}
disk_info {
 phys_stor[0] {
  name = "dscsi.s40"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 4; }
  unit = 0;
 }
 phys_stor[1] {
  name = "dscsi.s41"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 4; }
  unit = 1;
 }
 phys_stor[2] {
  name = "dscsi.s50"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 5; }
  unit = 0;
 }
 phys_stor[3] {
  name = "dscsi.s51"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 5; }
  unit = 1;
 }
 phys_stor[4] {
  name = "dscsi.s20"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 2; }
  unit = 0;
 }
 phys_stor[5] {
  name = "dscsi.s21"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 2; }
```

```
  unit = 1;
 }
 phys_stor[6] {
  name = "dscsi.s30"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 3; }
  unit = 0;
 }
 phys_stor[7] {
  name = "dscsi.s31"; disk_pe = "ospe_a";
  ptype = d_disk; stype = d_x500;
  iopath { ioc = 0; iop = 1; chan = 3; }
  unit = 1;
 }
 ram_stor[0] {
  name = "ramdev"; ptype = d_ram;
  length = 13288; length_unit = blocks;
 }
}
rootdev { name = "root"; minor = 2; }
swapdev { name = "swap"; minor = 22; }
boot_info {
 rev_level = 0;
 boot_info_size = 0;
 mf_name = "NULL";
 scx_ring = 0;
 scx_node = 15;
 scx_chan = 0;
 scx_dest_node = 1;
 archive_size = 0;
 mkpal_size = 0;
 config_size = 0;
 clock {
  month = 0;
  day = 0;
  year = 0;
  hour = 0;
  min = 0;
  sec = 0;
  tz = 0;
 }
 num_pes = 8;
 run_level = 0;
```

```
 boot_pe = 0x007;
 boot_options = 0x00;
 sn = 6501;
 hz = 75000000;
}
routes[0] {
 ringno = 0;
 io_contr_ppe = 0x10100;
 gigaring_node_addr = 15;
 path_type = primary;
 link_type = link_scx;
 intr_ps_name = "ospe_a";
 send_ps_name = "ospe_a";
}
mpp {
 pe[0:3] {
  flags = 0;
  pe_type = PE_TYPE_APP;
  actor_list_index = 1;
 }
 pe[4:6] {
  flags = 0;
  pe_type = PE_TYPE_CMD;
  actor_list_index = 0;
 }
 pe[7] {
  flags = 0;
  pe_index = 0x007; /* temporary */
  pe_type = PE_TYPE_OS;
  actor_list_index = 2;
  pe_name = "ospe_a";
 }
 pe_actors[0] {
  actor_name[0] = "kernel";
  actor_name[1] = "PM";
  actor_name[2] = "fsa";
  actor_name[3] = "em";
 }
 pe_actors[1] {
  actor_name[0] = "kernel";
  actor_name[1] = "PM";
  actor_name[2] = "fsa";
  actor_name[3] = "em";
```

```
 }
 pe_actors[2] {
  actor_name[0] = "kernel";
  actor_name[1] = "PM";
  actor_name[2] = "fsa";
  actor_name[3] = "log";
  actor_name[4] = "info";
  actor_name[5] = "em";
  actor_name[6] = "config";
  actor_name[7] = "packet";
  actor_name[8] = "disk";
  actor_name[9] = "tty";
  actor_name[10] = "netdev";
  actor_name[11] = "file";
  actor_name[12] = "socket";
  actor_name[13] = "alm";
  actor_name[14] = "grm";
  actor_name[15] = "GPM";
  actor_name[16] = "ipcm";
  actor_name[17] = "UP_INIT";
  actor_name[18] = "sio";
  actor_name[19] = "tape";
 }
}
audit_info {
 state = 0;
 all_nami = 0;
 all_rm = 0;
 all_valid = 0;
 audit = 1;
 chdir = 1;
 config = 1;
 crl = 0;
 dac = 1;
 discv = 1;
 filexfr = 1;
 physio_err = 0;
 ipnet = 1;
 jend = 1;
 jstart = 1;
 linkv = 1;
 mandv = 1;
 mkdirv = 1;
```

```
 netcf = 0;
 netwv = 1;
 nfs = 0;
 nqs = 0;
 nqscf = 0;
 path_track = 1;
 oper = 1;
 priv = 0;
 removev = 1;
 rmdirv = 1;
 secsys = 1;
 suid = 1;
 shutdown = 1;
 startup = 1;
 sulog = 1;
 tape = 0;
 user = 0;
 trust = 0;
 tchg = 1;
 panic = 1;
 maxsize = 8192000;
 dir = "/usr/adm/sl";
 file = "slogfile";
 fprefix = "s.";
}
var {
 nbuf = 2000;
 nbuf_fctr = 20;
 nhbuf_fctr = 4;
 nblk_fctr = 20;
 npbuf = 200;
 nasyn = 400;
 maxasyn = 35;
 nsu_asyn = 4;
 maxpipe = 20;
 maxrah = 8;
 ncrbuf = 4;
 link_max = 1000;
 nldmap = 250;
 nmnt = 250;
 nmt = 0;
 nldch = 2000;
 ldchcore = 0;
```

```
 nsidebuf = 48;
 npty = 128;
 nc1minraw = 20;
 nusers = 200;
 njobs = 300;
 nproc = 650;
 ncall = 650;
 nc_namlen = 15;
 nexecs = 4;
 ntext = 100;
 nclist = 1000;
 acnice = 1;
 cdlimit = 040000000;
 max_unlinked_bytes = 26214400;
 dmode = 1;
 maxup = 100;
 nmount = 75;
 nfile = 3000;
 ninode = 1500;
 nc1inode = 1500;
 nc_size = 1024;
 nquota = 100;
 nflocks = 100;
 user_mls = 1;
}
security {
 secure_scrub = 0;
 priv_su = 1;
 nfs_secure_export_ok = 1;
 nfs_remote_rw_ok = 1;
 minslevel = 0;
 dev_enforce_on = 0;
 secure_mac = 0;
 secure_net_options = 6;
 forced_socket = 0;
 random_pass_on = 0;
 pass_minsize = 8;
 pass_maxsize = 8;
 maxlogs = 5;
 logdelay = 0;
 delay_mult = 0;
 disable_time = 0; /* should be -1 */
 console_msg = 0;
```

```
disable_acct = 0;
maxslevel = 0;
sysvcomps = 0;
secure_system_console = "";
system_admin_console = "/dev/console";
secure_operator_console = "/dev/console";
declassify_disk = 0;
overwrite_count = 3;
declassify_pattern = 0;
sanitize_pattern = 0;
secure_pipe = 0;
fsetid_restrict = 0;
mls_obj_ranges = 0;
comparts[0] = "secadm";
comparts[1] = "sysadm";
comparts[2] = "sysops";
comparts[3] = "netadm";
comparts[4] = "unicos";
comparts[5] = "crayri";
comparts[6] = "comp24";
comparts[7] = "comp39";
comparts[8] = "comp63";
compbits[0] = 01;
compbits[1] = 02;
compbits[2] = 04;
compbits[3] = 010;
compbits[4] = 020;
compbits[5] = 040;
compbits[6] = 040000000;
compbits[7] = 04000000000000;
compbits[8] = 0400000000000000000000;
lvlname[0] = "level0";
lvlname[1] = "level1";
lvlname[2] = "level2";
lvlname[3] = "level3";
lvlname[4] = "level4";
lvlname[5] = "level5";
lvlname[6] = "level6";
lvlname[7] = "level7";
lvlname[8] = "level8";
lvlname[9] = "level9";
lvlname[10] = "level10";
lvlname[11] = "level11";
```

```
    lvlname[12] = "level12";
    lvlname[13] = "level10";
    lvlname[14] = "level14";
    lvlname[15] = "level15";
    lvlname[16] = "level16";
    lvlnums[0] = 0;
    lvlnums[1] = 1;
    lvlnums[2] = 2;
    lvlnums[3] = 3;
    lvlnums[4] = 4;
    lvlnums[5] = 5;
    lvlnums[6] = 6;
    lvlnums[7] = 7;
    lvlnums[8] = 8;
    lvlnums[9] = 9;
    lvlnums[10] = 10;
    lvlnums[11] = 11;
    lvlnums[12] = 12;
    lvlnums[13] = 13;
    lvlnums[14] = 14;
    lvlnums[15] = 15;
    lvlnums[16] = 16;
  }
}

END_CONFIG
```

The t3e_config File [11]

This chapter contains a description of the t3e_config file which is generated by the t3ems(8) diagnostic tool. This section is for information only, and is not intended to be a guide for editing the t3e_config file manually.

Caution: Always use the t3ems(8) diagnostic tool to edit the contents of the t3e_config file. The t3ems(8) diagnostic tool enforces the restrictions regarding which combinations of values are legal for a Cray T3E system.

The t3e_config file is divided into two sections: the first section contains all the system parameters and the second section contains information specific to each individual PE. The entries must appear one per line. Comments beginning with the # character can be inserted anywhere in the file.

11.1 t3e_config File Entries

A t3e_config file contains the following entries:

- SYSTEM_TYPE
- NUMBER_PES
- BOOT_NODE
- BOOT_PE
- LUT_MODE
- BOOT_DIAG_LEVEL
- PWHO_LWHO_LIST

The following sections describe the individual t3e_config file entries.

11.1.1 The SYSTEM_TYPE Entry

The SYSTEM_TYPE entry indicates whether the system is an air-cooled system or a liquid-cooled system. This entry also indicates whether the system is a system or a tester.

The value of the SYSTEM_TYPE entry is set with the System type button menu located in the T3EMS System Configuration window. The possible values that this entry can assume are described in the following table:

Value	System Type	Description
0	AC	air-cooled
1	LC	liquid-cooled
2	AC_TESTER	air-cooled tester
3	LC_TESTER	liquid-cooled tester

11.1.2 The NUMBER_PES Entry

The NUMBER_PES entry indicates the number of physical PEs in the system. This number includes the PEs which are logically disabled.

11.1.3 The BOOT_NODE Entry

The BOOT_NODE entry indicates the GigaRing node ID (in decimal) that the t3ems(8) diagnostic uses to boot the system. This value must be set to a GigaRing node that is physically located on the selected module. The t3ems(8) diagnostic sets this value automatically; however, t3ems(8) cannot verify this setting because there is no cross-reference between slot locations and GigaRing node ID values.

The BOOT_NODE entry is set from the Boot node ID field located in the T3EMS System Configuration window.

11.1.4 The BOOT_PE Entry

The BOOT_PE entry indicates the boot PE. The BOOT_PE value is the decimal representation of the physical PE number selected from the Boot PWho button menu in the T3EMS System Configuration window.

Currently, the BOOT_PE value must be specified in decimal. In a future release, it may be specified in octal (prefixed by a leading zero) or hexadecimal (prefixed by a leading 0x).

11.1.5 The LUT_MODE Entry

The LUT_MODE entry indicates the setting of the R_LUT_OVERRIDE_MODE bits of the NET_LUT_OVERRIDE register. It can assume the following possible values:

Value	Description
0	One node per address
1	Bit 0 = PWho X0 and bit 1 = PWho Y0
2	Bit 0 = PWho Y0 and bit 1 = PWho X0

The LUT_MODE entry is set from the LUT mode button menu in the T3EMS System Configuration window.

The LUT_MODE applies globally to the entire Cray T3E system rather than to a single PE. The default value for small Cray T3E systems (less than 256 PEs) is 0 and the default for large Cray T3E systems is 1. The t3ems(8) diagnostic sends the LUT mode to the PAL code at boot time.

For mode 1, bits 0 and 1 of the logical PE number must correspond to bit 2^0 of the X-dimension portion of the physical PE number and bit 2^0 of the Y-dimension portion of the physical PE number, respectively. For mode 2, bits 0 and 1 of the logical PE number must correspond to bit 2^0 of the Y-dimension portion of the physical PE number and bit 2^0 of the X-dimension portion of the physical PE number, respectively. When you enter a new logical PE number, you must enter a value within these restrictions.

11.1.6 The BOOT_DIAG_LEVEL Entry

The BOOT_DIAG_LEVEL entry is not currently used. In a future release, this entry will determine the level of testing which will be done during the t3e_hdw_boot phase of an OS boot. The value will consist of 8 bits, where the upper 4 bits specify the boundary scan level (valid values are 0 and 1) and the lower 4 bits represent the boot testing level. A value of 0 means no boot testing. See the "System Configuration" section of the *Cray T3E Offline Diagnostic Maintenance System Reference Manual*, publication HDM-135, for more information.

11.1.7 The PWHO_LWHO_LIST Entry

The PWHO_LWHO_LIST entry defines parameters for each PE present in the system. Each line defines one PE, with all values given in hexadecimal notation and separated by dashes. For any given system there must be exactly *NUMBER_PES* lines defined in this section or errors will result.

Each line of the PWHO_LWHO_LIST entry has 9 fields of information about each PE, as follows:

1. PWho — Contains the physical PE number.
2. LWho — Contains the logical PE number.
3. Enabled — Denotes whether or not a node is disabled. A node includes both the PE and the routing links. If the node is enabled, this field is set to 1, otherwise it is set to 0. This field should never be disabled for the boot PE.

   This field is set by selecting a PE from the list of PEs in the T3EMS PE Configuration window and then toggling the D button to enable or disable it.
4. Flipped — This field is not currently used.
5. CPU Type — This field is not currently used. In future releases, this field will be used to distinguish between the different types of CPU that a PE can have.
6. RAM Type — Denotes the type of DRAM used for this PE. This field is set by selecting a PE from the T3EMS PE Configuration window and then choosing a value from the DRAM type button menu. The possible values that this field can assume are as follows:

Value	Description (memory and chip organization)
0	60 ns
1	50 ns

   Values 2 through 7 are not currently used.
7. Memory Configuration — Denotes the types of memory configurations available to a PE. This field is set by selecting a

PE from the T3EMS PE Configuration window and then choosing a value from the Memory config button menu. The possible values that this field can assume are as follows:

Value	Description
0	64 Mbytes (1 Mx16/4K ref/cs0)
1	128 Mbytes (1 Mx16/4K ref/cs0&1)
2	256 Mbytes (2-1 Mx16/4K ref/cs0&1)
3	512 Mbytes (4-2 Mx8/2K ref/cs0&1)
4	256 Mbytes (4 Mx16/4K ref/cs0)
5	512 Mbytes (4 Mx16/4K ref/cs0&1)
6	1024 Mbytes (2-8 Mx8/4K ref/cs0&1)
7	2048 Mbytes (4-8 Mx8/4K ref/cs0&1)

8. Route Type

Denotes the types of routes used for this PE. This field is set by selecting a PE from the T3EMS PE Configuration window and then choosing a value from the Route type button menu. The possible values that this field can assume are as follows:

Value	Description
0	Short deterministic
1	Short adaptive
2	Long deterministic
3	Long adaptive
4	Pass 1 R-chip workaround

Values 5 through 7 are not currently used.

9. Links

The links field is a bitmask representing disabled/enabled links, I/O, PE and adaptive links present for a given node. Please note that there are interdependencies between the link

setting and which PEs are enabled, so simply setting a PE's enable to zero is not sufficient to disable a PE properly.

Use the `T3EMS PE Configuration` window for setting the links values on a given PE.

11.2 Sample `t3e_config` File

An example of a `t3e_config` file for an air-cooled system follows.

```
# Written by t3ems revision 1.0e on Fri May 30 06:45:47 1997
#
# t3ecfg library revision 1.4a
#
# This file should not be hand-edited!
#
# Begin system "Default"
SYSTEM_NAME     = Default
SYSTEM_TYPE     = 0
SYSTEM_CLASS    = 0
NUMBER_PES      = 8
SCAN_NODE       = 0xf
BOOT_NODE       = 0xf
BOOT_PE         = 0x10000
LUT_MODE        = 0
NETWORK_TYPE    = 0
#
#       pwho-lwho-enabled-flipped-cputype-ramtype-memcfg-rtetype-links
#
PWHO_LWHO_LIST  =
        000000-000-1-0-0-0-1-0-1002
        000001-001-1-0-0-0-1-0-1002
        000100-002-1-0-0-0-1-0-0210
        000101-003-1-0-0-0-1-0-0210
        010000-007-1-0-0-0-1-0-1002
        010001-004-1-0-0-0-1-0-1002
        010100-005-1-0-0-0-1-0-0210
        010101-006-1-0-0-0-1-0-0210
SUM = 3704770440
# End system
```

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y