CSL Parameter File [5]

Note: It is strongly recommended that you use the install tool to maintain your system configuration, rather than manually editing this file. For more information on the UNICOS installation and configuration menu system (ICMS), see the online help files and the *UNICOS System Configuration Using ICMS*, Cray Research publication SG–2412.

This chapter describes the configuration specification language (CSL) parameter file, which contains statements that specify hardware and software characteristics for your system.

Note: As of the UNICOS 9.2 release, ICMS is not used on UNICOS systems for installation or loading of the UNICOS operating system.

When the configuration is activated, a copy of the CSL parameter file is stored in `/etc/config/param`. For all Cray Research systems, the administrator must manually transfer the parameter file to the workstation.

5.1 CSL syntax

There are three classes of tokens that make up CSL:

- Identifiers
- Constants
- Operators, separators, and comments

White space (horizontal tabs, new lines, carriage returns, and spaces) separates individual tokens.

5.1.1 Identifiers

An *identifier* is a sequence of digits and letters that specify either special keywords (such as `CONFIG`) or specific objects (such as a physical device). The digits and characters can be enclosed by double quotes. The underscore (_) and dash (-) are interpreted as letters. Uppercase and lowercase letters are interpreted differently; that is, CSL identifiers are case-sensitive. There are no restrictions on the first character of an identifier.

For example, each of the following is a valid identifier:

```
tmp
STI
abc_d
29-A1-31
Dump
0x1246
"507"
_xyz
```

There are two classes of identifiers:

- *Keyword identifiers* have special meaning in CSL and cannot be used to name other things. For a list of reserved keywords, see Appendix A, page 105.
- *Object identifiers* name specific objects. Objects are divided into three classes:
  - Physical devices
  - Logical devices
  - Slices

Each class of object has its own name space. That is, each object in a given class must have a unique name, but objects in different classes can share the same name.

5.1.2 Constants

All constants are positive integers. An integer consists of 1 digit or a sequence of digits. If the first digit of a constant is (zero), the constant is interpreted as octal; otherwise, the constant is assumed to be decimal. The use of digits 8 or 9 in an octal constant causes an error.

The following are all examples of valid constants:

```
012345      12      44673      0003455
```

5.1.3 Operators, separators, and comments

The allowable operators and separators in CSL are as follows:

```
{          }          ;          ’
```

The meanings of these operators and separators depend on the context in which they are used.

To intersperse comments between objects, begin and end the comment text as follows:

```
/*  This is the comment text.  */
```

5.2 CSL usage

This section provides general information on CSL usage.

CSL statements are located in the CSL parameter file. CSL statements are used to extend or override the default configuration.

All statements in the CSL parameter file must be terminated by a semicolon.

5.2.1 The boot process

UNICOS processes CSL statements in order of appearance in the CSL parameter file at boot time.

When processing is finished, a copy of the CSL file is placed in the `/CONFIGURATION` file.

5.2.2 Configuration verification

You can verify configurations by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Verify the disk configuration
```

You can also use the user-level program `econfig`(8), which shows error messages for any errors in the configuration.

The `econfig` program accepts only valid CSL statements as input. It is recommended that you verify the CSL statements by using the `econfig` command before booting a new configuration to prevent receiving errors during CSL processing.

5.2.3 Error message syntax

If, while processing CSL statements, UNICOS detects a condition warranting your attention, a message describing the condition is written to the system console. Conditions warranting a message are divided into three levels of severity:

- Informational messages, which do not affect the boot process.
- Error messages, which inform you of a problem in statement syntax or configuration consistency that prevents the system from booting. Directive processing continues, but the system panics when statement processing completes.
- Panic messages, which inform you of a problem that prevents the completion of statement processing. The system panics immediately.

The syntax of a condition message is as follows:

```
severity:message_text, location
```

severity	Values are `INFO`, `ERROR`, and `PANIC`.
message_text	The message proper.
location	Problem location. The *location* is expressed as a line number in the CSL parameter file.

5.3 CSL parameter file sections

The statements in the CSL parameter file are organized in the file by functionality:

Section	Description
`dumpinfo`	IOS-E based system dump parameters
`gigaring`	GigaRing configuration parameters
`ios_e`	IOS-E configuration parameters
`mainframe`	Physical mainframe characteristics parameters

`unicos`	UNICOS kernel parameters
`filesystem`	Physical storage devices and file system layout parameters
`network`	Network parameters and device parameters
`mpp`	Massively parallel processing (MPP) parameters (IOS-E based systems only)
`revision`	Revision identifier parameters
`t90_config`	CRAY T90 configuration parameters

Caution: There are specific naming requirements for naming dump devices and dump locations. For information about these requirements, see *General UNICOS System Administration*, Cray Research publication SG–2301.

Note: The default parameter file contains sections that are I/O-specific and therefore will not be used by all systems. You should remove the unused sections from your mainframe's parameter file to avoid potential problems:

- For GigaRing based systems, remove the `dumpinfo` and `ios_e` sections.
- For IOS-E based systems, remove the `gigaring` section.

5.3.1 `dumpinfo` section

Caution: There are specific naming requirements for naming dump devices and dump locations. For information about these requirements, see *General UNICOS System Administration*, Cray Research publication SG–2301.

Note: This section does not apply to a GigaRing environment or to the CRAY J90 series.

The `dumpinfo` section of the CSL parameter file defines the types and ranges of memory to dump when invoking the OWS `dumpsys`(8) command.

The `dumpinfo` section is specified in the CSL parameter file using the following syntax:

```
dumpinfo {
          range_declaration_1;
          range_declaration_2;
                   .
                   .
                   .
   }
```

A range declaration is specified using the following syntax:

```
type range is start to stop units ;
```

type	Either `memory` or `SSD`, indicating central memory or SSD memory to be dumped.
start	Integer indicating the beginning location of a memory section to be dumped.
stop	Integer indicating the ending location of a memory section to be dumped.
units	Units in which the start and stop values are to be computed: `words`, `Mwords`, or `Gwords`.

Up to four declarations of each *type* are allowed in the `dumpinfo` section.

The following example shows a typical `dumpinfo` section of a CSL parameter file:

```
dumpinfo {
     memory range is 0 to 16 Mwords;
}
```

Note: If the first memory range specification is not large enough to contain the entire kernel space, the mainframe dump routine will override the site specification and try to dump the full kernel space when a system dump is performed. This can lead to a truncated dump if the dump device is not large enough to contain the expanded dump. A truncated dump will not include executing exchange packages, CPU registers, or user areas. This reduces the usefulness of the dump. To override this behavior, and to enforce the site-specified values, insert a memory range specification of `0 to 0` as the first range. Sites that have `LDCHCORE`, `PLCHCORE`, and/or a random access memory (RAM) disk configured commonly experience this, because the `LDCHCORE`, `PLCHCORE`, and RAM disk space is included within the kernel space.

For example:

```
dumpinfo {
     memory range is 0 to 0 Mwords;
     memory range is 0 to 16 Mwords;
}
```

5.3.2 `gigaring` section

Note: This section does not apply to IOS-E based systems.

GigaRing configuration in UNICOS is done in two places:

- Internal routing and path selection is done in the `gigaring` section of the parameter file.
- Physical channel designation is done in the `mainframe` section of the parameter file.

The `gigaring` section consists of two subsections:

- `gr_route`
- `gr_union`

5.3.2.1 The `gr_route` subsection

The `gr_route` subsection provides a means of internal routing and path selection known as *source routing*. Source routing chooses the channel used to route traffic to a given ring. Where more than one mainframe GigaRing channel exists on a ring, messages are routed in a round-robin fashion.

Routing information is declared in the `gr_route` subsection. For example:

```
gigaring {
        gr_route {
                ring 06 {
                        channel 024;
                }
                ring 07 {
                        channel 034;
                        channel 044;
                }
        }
}
```

There can be up to 8 routes per ring. By default, the first route declared is designated as the primary route. Valid ring numbers are decimal integers in the range 0 through 127. Valid node numbers are decimal integers in the range 1 through 63.

5.3.2.2 The `gr_union` subsection

A *GigaRing union* is a logical representation of a device that has more than one ring and node designation. For example, a CRAY SSD-T90 with four GigaRing connections (and therefore four ring and node addresses) can be represented by one `gr_union` ring and node address. This applies only to devices for which the mainframe acts as the master for the direct memory access (DMA) operation; in the current implementation, this is only the CRAY SSD-T90.

By convention, ring 0200 is reserved for GigaRing union devices. There is a maximum of 16 nodes (node numbers through 017) reserved for these devices. Devices that are configured as GigaRing union devices allow their device drivers to query the low-level master DMA driver for physical ring and node addresses. The device driver can then route the DMA requests by targeting one physical ring/node address. This is known as *destination routing*. DMA requests can be scheduled by targeting the least busy channel. The retrying of requests in error can be targeted to another destination.

The GigaRing union device allows for ease of configuration and backward compatibility with UNICOS device node methodology.

An example of a `gr_union` declaration is as follows:

```
gigaring {
        gr_union {
                ring 0200, node 01 {
                        ring 06, node 04;
                        ring 06, node 05;
                        ring 07, node 04;
                        ring 07, node 05;
                }
        }
}
```

In this example, a GigaRing union logical device designated as ring 0200, node 01, will be created and will consist of four physical destinations.

5.3.3 `ios_e` section

Note: This section does not apply to GigaRing based systems.

The `ios_e` section defines the topology of the IOS-E hardware. Specify the characteristics using the following menu selection:

```
Configure System
    ->IOS configuration
```

This section includes the following information:

- Number of clusters that constitute the IOS-E
- Number and type of I/O processors (IOPs) in each cluster
- High-speed (HISP) channel information (channel, target, and operating mode)
- Full OWS path names for each IOP binary

The `ios_e` section is specified in the CSL parameter file by the following statement:

`ios_e {` *list of cluster declarations* `}`

A cluster declaration is specified by the following statement:

```
cluster ordinal { list of cluster statements }
```

The following types of IOS cluster statements exist:

- IOP declarations
- HISP declarations
- IOP boot declarations

5.3.3.1 IOP declarations

IOP declarations are specified by the following statement:

```
iop;
```

The following are valid IOPs:

- `muxiop` or `miop`
- `eiop 0`
- `eiop 1`
- `eiop 2`
- `eiop 3`
- `eiop 4`

Note: `eiop 4`, `muxiop`, and `miop` are equivalent declarations except for the CRAY J90 series. For the CRAY J90 series, `eiop` indicates the VME controller type and may range from 0 through 50. See the *UNICOS Basic Administration Guide for CRAY J90 Series*, Cray Research publication SG–2416.

5.3.3.2 HISP declarations

Note: This section does not apply to the CRAY J90 series.

HISP declarations must be correct for the mainframe type on which they are declared, or system problems may occur. HISP declarations are specified by the following statement:

```
channel ordinal is hisp ordinal to chan_target , mode  chan_type ;
```

HISP channel targets are as follows:

- `mainframe`
- `SSD`

The only channel type is `c200d200` (or `C90`), which stands for 200 Mbyte control / 200 Mbyte data.

5.3.3.3 IOP boot declarations

Note: This section does not apply to the CRAY J90 series.

IOP boot declarations are specified by the following statement:

```
boot iop with "pathname" ;
```

If the *pathname* is fully resolved (to the root directory), it is used. If a partial path name is specified, it is appended to the OWS value for `ROOTDIR`, as specified in the OWS configuration file.

5.3.3.4 Example of `ios_e` section

The following example shows the `ios_e` section of a CSL parameter file:

```
ios_e {
        cluster 0 {
                muxiop; eiop 0; eiop 1; eiop 2; eiop 3;
                channel 010 is hisp 0 to mainframe, mode C90;
                channel 014 is hisp 1 to       SSD, mode c200d200;
                boot muxiop with "/home/ows1601/cri/os/ios/iopmux";
                boot eiop 0 with "/home/ows1601/cri/os/ios/eiop.comm";
                boot eiop 1 with "/home/ows1601/cri/os/ios/eiop.bmx";
                boot eiop 2 with "/home/ows1601/cri/os/ios/eiop.dca2";
                boot eiop 3 with "/home/ows1601/cri/os/ios/eiop.dca1";
        }
        cluster 1 {
                muxiop; eiop 0; eiop 1; eiop 2; eiop 3;
                channel 010 is hisp 0 to mainframe, mode C90;
                channel 014 is hisp 1 to       SSD, mode c200d200;
                boot muxiop with "/home/ows1601/cri/os/ios/iopmux";
                boot eiop 0 with "/home/ows1601/cri/os/ios/eiop.dca1";
                boot eiop 1 with "/home/ows1601/cri/os/ios/eiop.dca1";
                boot eiop 2 with "/home/ows1601/cri/os/ios/eiop.dca2";
                boot eiop 3 with "/home/ows1601/cri/os/ios/eiop.hippi";
        }
}
```

5.3.4 `mainframe` section

The `mainframe` section defines the following hardware parameters:

- Number of CPUs
- Number of mainframe cluster registers
- Size of the mainframe memory
- Spare chip configuration (CRAY C90 series only)
- Channel information
  - Physical channel for a GigaRing environment
  - Low-speed channel information (channel and target) for IOS-E based systems
  - VHISP channel information for IOS-E based systems

Set these parameters by using the following menu selection for IOS-E based systems:

```
Configure System
    ->Mainframe hardware configuration
```

The `mainframe` section is specified in the CSL parameter file by the following statement:

```
mainframe { list of hardware definitions }
```

5.3.4.1 Common parameters

The following parameters are common to GigaRing based systems and IOS-E based systems:

- Number of CPUs
- Number of mainframe cluster registers
- Size of memory

5.3.4.1.1 Number of CPUs

The number of CPUs is specified by the following statement:

```
value cpus ;
```

value is the physical number of CPUs in the machine. Specifying a smaller value will cause UNICOS to use only that many CPUs, starting at CPU 0.

5.3.4.1.2 Number of mainframe cluster registers

The number of mainframe cluster registers is specified by the following statement:

```
value clusters ;
```

value is the number of clusters. If this is not specified, it defaults to 1 greater than the `cpus` value.

5.3.4.1.3 Size of memory

The mainframe memory size is specified by the following statement:

```
value units memory[,mem_halving] ;
```

units may be either `words` or `Mwords`. Typically, *value* is set to the physical amount of memory in the machine, but it can be set to a smaller value. This may be useful when experiencing memory problems.

The *mem_halving* parameter is for the CRAY C90 series only. It specifies the hardware memory configuration. Values can be as follows:

- `lower half`
- `upper half`
- `both halves`

Your customer engineer can tell you the memory configuration for your machine.

5.3.4.2 Parameters for GigaRing based systems only - channel declarations

The physical channel configuration declares a physical channel to be a GigaRing channel. It creates a GigaRing port by assigning a ring number and node number to a given mainframe channel number.

```
channel ordinal is gigaring to ring ring_number, node node_number ;
```

ordinal is the channel number. *ring_number* must be an integer in the range through 127. *node_number* must be an integer in the range 1 through 63. For every channel entry in the `gr_route` and `gr_union` sections, there must be a corresponding `channel` entry in the `mainframe` section.

At UNICOS boot time, a `cnode` structure is declared to represent each GigaRing port. The ring and node numbers will be read and verified from the GigaRing node, or, in the case of a direct connect, be set according to the ring and node numbers specified.

The following channel numbers are valid for CRAY J90se systems:

```
024         064
034         074
044         0104
054         0114
```

The following channel numbers are valid for CRAY T90 GigaRing based systems (all even octal integers from 0100 through 0176):

```
0100        0120        0140        0160
0102        0122        0142        0162
0104        0124        0144        0164
0106        0126        0146        0166
0110        0130        0150        0170
0112        0132        0152        0172
0114        0134        0154        0174
0116        0136        0156        0176
```

Table 18 shows the channels for I/O module 0.

Table 18. I/O module 0 channels

Erred packet queue 0	Erred packet queue 1	Incoming packet queue 0	Incoming packet queue 1	Outgoing packet queue	DMA TIB/TCB buffer	Ring number
0100	0101	0200	0201	0300	0301	GR 0
0102	0103	0202	0203	0302	0303	GR 1
0104	0105	0204	0205	0304	0305	GR 2
0106	0107	0206	0207	0306	0307	GR 3
0110	0111	0210	0211	0310	0311	GR 4
0112	0113	0212	0213	0312	0313	GR 5
0114	0115	0214	0215	0314	0315	GR 6
0116	0117	0216	0217	0316	0317	GR 7

Table 19 shows the channel number ranges for a CRAY T932 configuration of four IO2 modules.

Table 19. Example of complete channel assignments

Erred packet queue 0	Erred packet queue 1	Incoming packet queue 0	Incoming packet queue 1	Outgoing packet queue	DMA TIB/TCB buffer	IO2 module number
0100-0116	0101-0117	0200-0216	0201-0217	0300-0316	0301-0317	IO2 - 0
0120-0136	0121-0137	0220-0236	0221-0237	0320-0336	0321-0337	IO2 - 1
0140-0136	0141-0157	0240-0256	0241-0257	0340-0356	0341-0357	IO2 - 2
0160-0176	0161-0177	0260-0276	0261-0277	0360-0376	0361-0377	IO2 - 3

5.3.4.3 Parameters for IOS-E based systems only

The following parameters are for IOS-E based systems only:

- Channel declarations
- Spare chip configuration (CRAY C90 only)

5.3.4.3.1 Channel declarations

Channel declarations are either for a low-speed channel pair or a VHISP channel. A channel declaration is specified by the following statement:

```
channel ordinal is channel_type to channel_target ;
```

ordinal is the channel number, for which the preferred format is octal. *channel_type* is either `lowspeed` or `VHISP`. A low-speed *channel_target* is the cluster ordinal or `pseudo TCP`. The VHISP *channel_target* is `SSD`. You must specify `pseudo TCP` for TCP/IP to be operational.

5.3.4.3.2 Spare chip configuration for CRAY C90 series

For the CRAY C90 series, specify the path name to the spare chip configuration file on the OWS by using the following statement:

```
configure C90 spares with "pathname" ;
```

If the *pathname* is fully resolved (to the root directory), it is used. If a partial path name is specified, it is appended to the OWS's value for `ROOTDIR`, as specified in the OWS configuration file. Your customer engineer can tell you if your CRAY C90 system has the memory chip sparing capability.

5.3.4.4 Example of the `mainframe` section for a GigaRing based system

The following shows an example of the `mainframe` section of the CSL parameter file for a GigaRing based system:

```
mainframe {
        2 cpus;
        16 Mwords memory;
        channel 024 is gigaring to ring 6, node 1;
        channel 034 is gigaring to ring 7, node 5;
        channel 044 is gigaring to ring 7, node 6;
}
```

5.3.4.5 Example of the `mainframe` section for an IOS-E based system

The following shows an example of the `mainframe` section of the CSL parameter file in an IOS-E based system:

```
mainframe {
        2 cpus;
        32 Mwords memory;
        channel 16 is lowspeed to cluster 0;
        channel 18 is lowspeed to cluster 1;
        channel  1 is vhisp  0 to SSD;
}
```

5.3.5 unicos section

The `unicos` section sets certain tunable parameters. Set these parameters by using the following menu selection:

```
Configure System
    ->UNICOS Kernel Configuration
```

The `unicos` section is specified in the CSL parameter file by the following statement:

`UNICOS {` *list of tunable parameters* `} ;`

A UNICOS tunable parameter is specified by the following statement:

```
value parameter ;
```

All systems have the same tunable parameters for online tapes, table sizes, and maximum limits. Table 20 through Table 22 show these parameters.

Table 20. Online tape parameters

Parameter	Description
`TAPE_MAX_CONF_UP`	Maximum number of tape devices that can be configured up at the same time.
`TAPE_MAX_DEV`	Maximum number of tape devices.
`TAPE_MAX_PER_DEV`	Maximum number of bytes allocated per tape device.

Table 21. Table size parameters

Parameter	Description
`LDCHCORE`	Memory clicks reserved for ldcache blocks assigned as type `MEM`.
`NLDCH`	Number of ldcache headers.
`NPBUF`	Number of physical I/O buffers.

Table 22. Maximum limits parameters

Parameter	Description
`GUESTMAX`	Maximum number of guest systems.
`NBUF`	Number of buffer headers.
`NGRT`	Maximum number of guest resource table entries.

GigaRing based systems and IOS-E based systems have common and unique disk parameters. Table 23 through Table 25 show these parameters.

Table 23. Disk parameters (common)

Parameter	Description
`LDDMAX`	Maximum number of logical disk devices (LDDs). This value also limits the minor number for this type. The maximum minor number is `LDDMAX` –1.
`MDDSLMAX`	Maximum number of mirrored disk device (MDD) slices. This value also limits the minor number for this type. The maximum minor number is `MDDSLMAX` –1.
`PDDMAX`	Maximum number of physical disk devices (PDDs).
`PDDSLMAX`	Maximum number of PDD slices. This value also limits the minor number for this type. The maximum minor number is `PDDSLMAX` –1.
`RDDSLMAX`	Maximum number of RAM disk device (RDD) slices. This value also limits the minor number for this type. The maximum minor number is `RDDSLMAX` –1.
`SDDSLMAX`	Maximum number of striped disk device (SDD) slices. This value also limits the minor number for this type. The maximum minor number is `SDDSLMAX` –1.

Table 24. Disk parameters (GigaRing based systems only)

Parameter	Description
`SSDTMAX`	Maximum number of SSD-T90 devices. The value must be an integer in the range 0 through 8. 0 is the default.
`SSDTSLMAX`	Maximum number of slices that can be allocated to the SSD-T90 device. This value must be an integer in the range 0 through 2^{15} –1, depending upon the value set for `SSDTMAX`. The default is 32. This value also limits the minor number for this type. The maximum minor number is `SSDTSLMAX` –1.

Parameter	Description
`XDDMAX`	Maximum number of physical devices. The default is 32.
`XDDSLMAX`	Maximum number of physical slices. The default is 256.

Table 25. Disk parameters (IOS-E based systems only)

Parameter	Description
`HDDMAX`	Maximum number of HIPPI disk devices (HDDs).
`HDDSLMAX`	Maximum number of HDD slices. This value also limits the minor number for this type. The maximum minor number is `HDDSLMAX` 1.

Table 26 shows the valid unit bit and range numbers for IONs.

Table 26. ION unit bit and range numbers (GigaRing based systems only)

ION type	Bits	Range
FCN	Loop ID 0–7	0–127
HPN	Facility bits 0–8	0–127
IPN	Unit bits on a daisy chain 0–2	0–7
MPN	Device ID 0-7 Logical unit number 0–7	0–14
RAID	RAID partition bits 9–15	0–127

5.3.5.1 Example for a GigaRing based system

The following is an example `unicos` section for a GigaRing based system:

```
unicos {
        2480    NBUF;                   /* System buffers */
        100     NLDCH;                  /* Ldcache headers */
        2000    LDCHCORE;               /* Ldcache memory */
        150     LDDMAX;                 /* Max. number of LDD devices */
        150     PDDMAX;                 /* Max. number of PDD devices */
        256     PDDSLMAX;               /* Max. number of DISK slices */
        32      XDDMAX                  /* Max. number of xdisk devices */
        256     XDDSLMAX                /* Max. number of xdisk slices */
        8       SDDSLMAX;               /* Max. number of SDD slices */
        8       MDDSLMAX;               /* Max. number of MDD slices */
        4       RDDSLMAX;               /* Max. number of RAM slices */
        30      TAPE_MAX_CONF_UP;       /* Max. number of tapes configured */
        65536   TAPE_MAX_PER_DEV;       /* Max. tape buffer size */
}
```

5.3.5.2 Additional parameters for a CRAY SSD-T90 system

The following is an example of additional parameters required for a CRAY SSD-T90 system:

```
unicos {
        2         SSDTMAX;
        16        SSDTSLMAX;
}
```

5.3.5.3 Example for an IOS-E based system

The following is an example `unicos` section for an IOS-E based system:

```
unicos {
        2480    NBUF;                   /* System buffers */
        100     NLDCH;                  /* Ldcache headers */
        2000    LDCHCORE;               /* Ldcache memory */
        150     LDDMAX;                 /* Max. number of LDD devices */
        150     PDDMAX;                 /* Max. number of PDD devices */
        256     PDDSLMAX;               /* Max. number of DISK slices */
        8       SDDSLMAX;               /* Max. number of SDD slices */
        8       MDDSLMAX;               /* Max. number of MDD slices */
        4       RDDSLMAX;               /* Max. number of RAM slices */
        4       SSDDSLMAX;              /* Max. number of SSD slices */
        30      TAPE_MAX_CONF_UP;       /* Max. number of tapes configured */
        65536   TAPE_MAX_PER_DEV;       /* Max. tape buffer size */
}
```

5.3.6 `filesystem` section

The `filesystem` section includes the following:

- Description of the physical devices in the system:
  - xdisks, disks, RAM, and SSDT (CRAY SSD-T90) for GigaRing based systems
  - disks, RAM, and SSD for IOS-E based systems
- Description of the device nodes in the system:
  - XDD, QDD, PDD, RDD, MDD, HDD, and SDD for GigaRing based systems
  - PDD, LDD, RDD, MDD, HDD, and SDD for IOS-E based systems
- Identification of the root, swap, SDS, and dump devices
- Description of the CRAY SSD-T90 (GigaRing based systems only)

Set these parameters by using the following menu selection:

```
Configure System
    ->Disk configuration
```

For information on striping file systems, see *General UNICOS System Administration*, Cray Research publication SG–2301.

The beginning of the `filesystem` section is indicated by the following line in the CSL parameter file:

```
filesystem {
```

The following sections describe each portion of the `filesystem` section of the parameter file. For more detailed information on physical device specification, see *General UNICOS System Administration*, Cray Research publication SG–2301.

Set these parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Physical Devices
```

5.3.6.1 Physical device definition

The following sections describe the definition of physical storage devices for GigaRing based systems and IOS-E based systems. Table 27 summaries the disk types and names.

Table 27. Disk type and name

Disk type	Disk name
IPI-2	`DD3`, `DD4`, `DD5I`, `DDAS2`, `DDESDI`, `DDIMEM`, `RAM`, `RD-1`, `DA60`, `DA62`, `DA301`, `DA302`, `DD40`, `DD41`, `DD42`, `DD49`, `DD50`, `DD60`, `DD61`, `DD62`, `DD301`, `DD302`, `HD16`, `HD32`, `HD64`, `RAM`, `SSD`,
SCSI	`DD314`, `DD318`, `DD501`, `DD5S`, `DD6S`, `DD7S`, `DD_U`
Fibre	`DD000`, `DD304`, `DD308`, `DD309`, `DD310`
HIPPI	`HD16`, `HD32`, `HD64`
Special	`RAM`, `SSD`, `SSDT`

5.3.6.1.1 Slice definitions for physical storage devices

Each physical storage device can be segmented into one or more pieces; each piece is a *slice*. For each slice, a *minor device number*, a starting device address, and the slice length must be specified. The starting device address and the slice length can be specified in units of sectors or blocks. A *block* is defined to be 4096 bytes (512 64-bit words). A *sector* is an integer multiple of some number of blocks that varies with different physical storage devices. The ratio of the device's sector size to the size of a block is defined as one `iounit`. For example, a physical disk device may be formatted with a sector size of 16,384 bytes, giving it an `iounit` value of 4.

Generally, the start and length of a `disk` slice is specified in units of `sectors`. If the unit value is `blocks`, the `start` and `length` values must be integer multiples of the `iounit` value. The `iounit` value for a physical disk storage device is determined by the device `type` designator in the physical `disk` declaration.

Generally, the start and length of an `xdisk` slice are specified in units of `sectors`. If `blocks`, the `start` and `length` values must be integer multiples of the `iounit` value. The `iounit` value for a physical disk storage device is determined by the `iounit` designator in the physical `xdisk` declaration. In the case of an `xdisk` where no `iounit` is declared, the `iounit` value defaults to 1.

The start and length of a slice of a RAM disk are specified in `blocks`.

The start and length of an `ssd` slice can be specified in units of either `blocks` or `sectors`. If `sectors`, the `iounit` value of the SSD is determined by the optional `type` designator in the physical SSD declaration. If a `type` is not specified, the `iounit` value defaults to 1.

The start and length of an `ssdt` slice can be either `blocks` or `sectors`. If `sectors`, are used, the `iounit` of the CRAY SSD-T90 device is determined by the optional `iounit` designator in the physical CRAY SSD-T90 device declaration. If an `iounit` designator is not specified, the `iounit` value defaults to 1.

The following example shows the two forms for slice configuration:

```
slice_type  slice_name {
            minor minor_number;
            sector starting_sector_number;
            length length_in_sectors sectors;
   }
```

```
slice_type  slice_name {
            minor minor_number;
            block starting_block_number
            length length_in_blocks blocks;
   }
```

Device minor numbers must be unique among all slices of a given storage device type. They must be greater than 0, and less than the maximum number of slices specified in the `unicos` section of the parameter file.

Table 28 shows the preferred and optional start and length units for the various physical storage device types, as well as the parameter in the `unicos` section that determines the maximum value allowable for the minor device number.

Table 28. Start and length units for physical storage device types

Physical storage device type	Preferred units	Optional units	Maximum minor number –1
`disk`	`sectors`	`blocks`	`PDDSLMAX`
`xdisk`	`sectors`	`blocks`	`XDDSLMAX`
`hippi_disk`	`sectors`	`blocks`	`HDDSLMAX`
`RAM`	`blocks`	(none)	`RDDSLMAX`
`SSD`	`blocks`	`sectors`	`SSDDSLMAX`
`SSD-T90`	`blocks`	`sectors`	`SSDTSLMAX`

5.3.6.1.2 Physical device definition for GigaRing based systems

The following types of physical storage devices are available for GigaRing based systems:

- Random access memory (RAM)
- Physical storage devices (disk and xdisk)
- Solid-state storage device for a CRAY T90 system (CRAY SSD-T90)

Table 29 summarizes disk information for GigaRing based systems.

Table 29. Disk information (GigaRing based systems only)

Disk type	ION	PCA type	Driver	Node residence	Major number	CSL type	`mkspice` value
IPI-2	IPN	SPN	`qdd`	`/dev/pdd`	`dev_qdd`	`disk`	`YES`
SCSI	MPN	MPN	`xdd`	`/dev/xdd`	`dev_xdd`	`xdisk`	`NO`
Fibre	FCN	SPN	`xdd`	`/dev/xdd`	`dev_xdd`	`xdisk`	`NO`
HIPPI	HPN	MPN	`xdd`	`/dev/xdd`	`dev_xdd`	`xdisk`	`NO`

The RAM device definition has the following format (which is identical to that in an IOS-E based system):

```
RAM ram_name
     { length length_number units ;
     pdd pdd_slice_name
         { minor minor_number
           block starting_block_number
           length length_in_blocks blocks
     }
```

ram_name	Name of the RAM, which must be unique among all devices.
length_number	Size of the RAM, specified in *units*.
units	One of the following: `blocks`, `words`, or `Mwords`.
pdd_slice_name	Name of the slice.
minor_number	Minor number of the slice, which must be unique across the device type.
starting_block_number	Block number where the slice starts.
length_in_blocks	Length of the slice in blocks.

The physical storage device definition has the following format in a GigaRing based system:

```
disk device_name {
    type type;
    iopath {
          ring ring_number;
          node node_number;
          channel channel_number;
  }
    unit disk_unit_number;
     pdd pdd_slice_name {
          minor minor_number;
          sector starting_sector_number;
          length length_in_sectors sectors;
          }
    }
```

device_name	Name of the physical storage device, which must be unique among all devices.
type	Type of the physical storage device.
ring_number	Number of the I/O path ring.
node_number	Number of the I/O path node.
channel_number	Number of the I/O path channel. The channel specified is the channel in the peripheral channel adapter (PCA). You must include a leading 0 to specify the channel number in octal form.
disk_unit_number	Number of the disk. For disk devices that can be daisy chained, the unit number specifies the physical unit number of the device. It is recommended that start and length for disk devices be expressed in sectors.
pdd_slice_name	Name of the slice, which must be unique among all slices for all devices.
minor_number	Minor number of the slice, which must be unique across the device type.
starting_sector_number	Starting sector number of the slice.
length_in_sectors	Length of the slice in sectors.

The SSDT definition applies only to GigaRing based systems. It has the following format:

```
ssdt ssdt_name {
          iounit iounit_value;
          tmtype tmtype_number;
          [lunit lunit_number;]
          iopath {
                ring ring_number;
                node node_number;
          }
          piopaths piopaths_bitmask;
          xdd xdd_name {
                minor minor_number;
                block starting_block_number;
                length length_in_blocks blocks;
          }
  }
```

ssdt_name	Name of the CRAY SSD-T90, which must be unique among all devices.
iounit_value	Block size in 4096–byte units.
tmtype_number	Target memory type, which determines the memory address and configuration characteristics for a given CRAY SSD-T90. This value must be an integer in the range through 255. The default is 1.
lunit_number	The logical unit number of the CRAY SSD-T90; this is only used if there is more than one CRAY SSD-T90 device connected to the system.
ring_number	GigaRing ring number for the CRAY SSD-T90 device. (This can be a logical ring/node address in the form of a `gr_union` device; see Section 5.3.2.2, page 40.)
node_number	GigaRing node number for the CRAY SSD-T90 device. (This can be a logical ring/node address in the form of a `gr_union` device; see Section 5.3.2.2, page 40.)
piopaths_bitmask	(Optional)A bit mask representing the number of physical paths (GigaRing channels) used to split a single request across multiple CRAY SSD-T90 connections to a CRAY T90 mainframe. Each bit in the bit mask is a possible path:

- 03 represents 2 paths, which is standard for 512-Mword CRAY SSD-T90 devices
- 017 represents 4 paths, which is standard for 1024-Mword CRAY SSD-T90 devices

	Using this parameter can increase the bandwidth of individual large requests but will result in higher system overhead and may decrease overall CRAY SSD-T90 throughput. The `piopaths` parameter is typically used for specific applications (such as NASTRAN) that need the additional bandwidth and cannot use asynchronous requests.
xdd_name	Name of the slice, which must be unique among all slices for all devices.
minor_number	Minor number of the slice, which must be unique across the device type.
starting_block_number	Starting block number of the slice.
length_in_blocks	Length of the slice in blocks.

The xdisk definition applies only to GigaRing based systems. It has the following format:

```
xdisk xdisk_name {
     iounit iounit_value;
     iopath {
            ring ring_number;
            node node_number;
            channel channel_number;
            }
     unit disk_unit_number
     xdd xdd_slice_name {
            minor minor_number;
            sector starting_sector_number;
            length length_in_sectors;
            }
     }
```

xdisk_name	Name of the physical storage device, which must be unique among all devices.

iounit_value	Sector size in 4096–byte I/O units.
ring_number	GigaRing ring number of the peripheral channel adapter (PCA).
node_number	GigaRing ring node number of the PCA.
channel_number	Number of the I/O path channel. The channel specified is the channel in the PCA. You must include a leading 0 to specify the channel number in octal form.
disk_unit_number	Device unit number.
xdd_slice_name	Name of the slice, which must be unique among all slices for all devices.
minor_number	Minor number of the slice, which must be unique across the device type.
starting_sector_number	Starting sector number of the slice.
length_in_sectors	Length of the slice in sectors.

5.3.6.1.3 Physical device definition for IOS-E based systems

The following types of physical devices are available for IOS-E based systems:

- Random access memory (RAM)
- Physical storage devices
- Solid-state storage device (SSD)

The RAM device definition has the following format (which is identical to that in a GigaRing based system):

```
RAM ram_name
     { length length_number units ;
     pdd pdd_slice_name
         { minor minor_number
           block starting_block_number
           length length_in_blocks blocks
     }
```

ram_name	Name of the RAM, which must be unique among all devices.
length_number	Size of the RAM, specified in *units*.
units	One of the following: `blocks`, `words`, or `Mwords`.
pdd_slice_name	Name of the slice.
minor_number	Minor number of the slice, which must be unique across the device type.
starting_block_number	Block number where the slice starts.
length_in_blocks	Length of the slice in blocks.

The physical storage device definition has the following format for an IOS-E based system:

```
disk device_name {
     type type;
     iopath {
            cluster cluster_number;
            eiop eiop_number;
            channel channel_number;
     }
     unit disk_unit_number;
     pdd pdd_slice_name {
            minor minor_number;
            sector starting_sector_number;
            length length_in_sectors sector;
            }
     }
```

device_name	Name of the physical storage device, which must be unique among all devices.
type	Type of the physical storage device.
cluster_number	Number of the I/O cluster. **Note:** For the CRAY J90 series, `cluster` specifies the VME IOS, and `eiop` specifies the controller within the VME IOS. For more information, see the *UNICOS Basic Administration Guide for CRAY J90 Series*, Cray Research publication SG–2416.
eiop_number	Number of the EIOP I/O processor.
channel_number	Number of the I/O path channel. You must include a leading 0 to specify the channel number in octal form.
disk_unit_number	Number of the disk. For disk devices that can be daisy chained, the unit number specifies the physical unit number of the device. It is recommended that start and length for disk devices be expressed in sectors.
pdd_slice_name	Name of the slice.

minor_number	Minor number of the slice, which must be unique across the device type.
starting_sector_number	Starting sector number of the slice.
length_in_sectors	Length of the slice in sectors.

The solid-state storage device (SSD) definition has the following format:

```
SSD ssd_name
     { length length_number units ;
     pdd pdd_slice_name {
          minor minor_number;
          block starting_block_number;
          length length_in_blocks blocks;
          }
     }
```

ssd_name	Name of the SSD, which must be unique among all devices.
length_number	Size of the SSD.
units	One of the following: `blocks`, `Mwords`, or `sectors`.
pdd_slice_name	Name of the slice, which must be unique among all slices for all devices.
minor_number	Minor number of the slice, which must be unique across the device type.
starting_block_number	Starting block number of the slice.
length_in_blocks	Length of the slice in blocks.

5.3.6.2 Device node definition

The following device nodes can be defined in the `filesystem` section of the CSL parameter file:

Device type	Description
`pdd`	Physical device for use with the CSL type `disk`.
`ldd`	Logical device.
`sdd`	Striped device.

`mdd`	Mirrored device.
`qdd`	Physical device for GigaRing based systems; used to divide `xdisk` entries in the `filesystem` section.
`xdd`	Physical disk device for GigaRing based systems for use with the CSL type `xdisk`.

The only limitation is that any slice used in a node definition must have been defined in a physical storage device definition. For more information on mirrored and striped devices, see *General UNICOS System Administration*, Cray Research publication SG–2301.

Set the QDD and PDD parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Physical Device Slices (/dev/pdd entries)
```

Set the LDD parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Logical Devices (/dev/dsk entries)
```

Set the SDD parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Striped Devices (/dev/sdd entries)
```

Set the MDD parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Mirrored Devices (/dev/mdd entries)
```

Set the XDD parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Physical Device Slices on GigaRing Systems
        (/dev/xdd entries)
```

Each node definition has the following syntax:

```
node_type name {
        minor number;
        device slice;
```

The node type and device are one of the device types defined in the previous list. The name is site-configurable. The minor number is required and must be unique across the device type. The slice is a name of a slice (or slices or other device node definition) previously defined in your CSL parameter file.

5.3.6.3 CRAY SSD-T90 description

The CRAY SSD-T90 configuration reflects the explicit nature of the GigaRing, as opposed to the implicit VHISP form used for IOS-E based systems. A CRAY SSD-T90 device can have either a physical GigaRing `iopath` or a GigaRing union `iopath`.

Note: The CRAY SSD-T90 has an `iopath` designator (without a channel number) in its declaration.

The `lunit` (logical unit) parameter is the device ordinal that will be assigned to this physical CRAY SSD-T90. With one CRAY SSD-T90, the `lunit` parameter is optional and defaults to 0. If more than one CRAY SSD-T90 is designated, `lunit` is required, must be unique among SSDs, and must be smaller than the maximum number of CRAY SSD-T90 devices. The maximum number of CRAY SSD-T90 devices is designated by the `SSDTMAX` parameter in the `unicos` section of the parameter file. See Section 5.3.6.1, page 55, for the format.

The `tmtype` parameter is the target memory type associated with the CRAY SSD-T90. The target memory type determines memory address and configuration characteristics for a given CRAY SSD-T90. Table 30 shows the `tmtype` values.

Table 30. Target memory type values

`tmtype` value	Description
8	An 8–processor system
9	A 16–processor system

5.3.6.4 Root, swap, and secondary data segment (SDS) devices

The statements for the root, swap, and SDS devices have the following syntax in the `filesystem` section of the CSL parameter file for GigaRing based systems:

```
rootdev is ldd name
  swapdev is ldd name;
  sdsdev is xdd name;
  dmpdev is ldd name;
```

`sdsdev` must be an SSD or CRAY SSD-T90 slice; the others must be LDD definitions.

These statements have the following syntax for IOS-E based systems:

```
rootdev is ldd name;
  swapdev is ldd name;
  sdsdev  is pdd name;
  dmpdev is ldd name;
```

Set these parameters by using the following menu selection:

```
Configure System
    ->Disk Configuration
        ->Special System Device Definitions
```

5.3.6.5 Example of the `filesystem` section containing a RAM file system

The following example shows the `filesystem` section of a working CSL parameter file containing a RAM file system:

Note: Additional CSL tags are required in the `unicos` section. See Section 5.3.5, page 49.

```
filesystem {
   RAM ramdev    {length 10240 blocks;
       pdd ram          {minor   3; block        0; length    10240 blocks;}
   }
}
```

5.3.6.6 Example of the `filesystem` section for a GigaRing based system

The following example shows the `filesystem` section of a working CSL parameter file for a GigaRing based system:

Note: Additional CSL tags are required in the `unicos` section. See Section 5.3.5, page 49.

```
filesystem {
    /* Physical device configuration */
    xdisk d04026.3 { iounit 1;
        iopath { ring 04; node 02; channel 06; } unit  03;
        pdd 04026.3_usr_i { minor 231; sector 0;      length 444864 sectors; }
        pdd 04026.3_ccn   { minor 232; sector 444864; length 222432 sectors; }
    }
    xdisk d03020.0 { iounit 1;
        iopath { ring 03; node 02; channel 0; } unit   0;
        xdd mpn.s400  { minor 17; sector 0;      length 102000 sectors; }
        xdd mpn.roote { minor 18; sector 102000; length 250000 sectors; }
        xdd mpn.usre  { minor 19; sector 352000; length 250000 sectors; }
    }
    /* HPN Device */
    xdisk d257.200.78.223 { iounit 22;
        iopath { ring 02; node 05; channel 07; }
        unit  40136; ifield 0337;
        xdd hpn.1     { minor 23; block 0; length 250000 blocks; }
    }
    /* Logical device configuration */
    ldd usr_i  { minor 86; xdd 04026.3_usr_i; }
    ldd ccn    { minor 40; xdd 04026.3_ccn  ; }
    ldd root_e { minor 17; xdd mpn.roote    ; }
    ldd usr_e  { minor 30; xdd mpn.usre     ; }
    ldd swap   { minor  2; xdd mpn.s400     ; }
    ldd hpn    { minor 10; xdd hpn.1        ; }
    rootdev is ldd root_e;
    swapdev is ldd swap;
}
```

5.3.6.7 Example of the `filesystem` section for a GigaRing based system with disk devices configured for third-party I/O

The following example shows the `filesystem` section of a working CSL parameter file for a GigaRing based system with disk devices configured for third-party I/O.

Note: Additional CSL tags are required in the `unicos` section. See Section 5.3.5, page 49.

```
filesystem {
    /* Physical device configuration */
    xdisk d04026.3 { iounit 1;
        iopath { ring 04; node 02; channel 06; } unit  03;
        pdd 04026.3_usr_i { minor 231; sector 0;      length 444864 sectors; }
        pdd 04026.3_ccn   { minor 232; sector 444864; length 222432 sectors; }
    }
    xdisk d03020.0 { iounit 1;
        iopath { ring 03; node 02; channel 0; } unit   0;
        xdd mpn.s400  { minor  17; sector 0;      length 102000 sectors; }
        xdd mpn.roote { minor  18; sector 102000; length 250000 sectors; }
        xdd mpn.usre  { minor  19; sector 352000; length 250000 sectors; }
        xdd mpn.dump  { minor 136; sector 807360; length 100000 sectors; }
    }
    /* HPN Device */
    xdisk d257.200.78.223 { iounit 22;
        iopath { ring 02; node 05; channel 07; }
        unit  40136; ifield 0337;
        xdd hpn.1     { minor 23; block 0; length 250000 blocks; }
    }
    /* Logical device configuration */
    ldd usr_i  { minor  86; xdd 04026.3_usr_i ; }
    ldd ccn    { minor  40; xdd 04026.3_ccn   ; }
    ldd root_e { minor  17; xdd mpn.roote     ; }
    ldd usr_e  { minor  30; xdd mpn.usre      ; }
    ldd swap   { minor   2; xdd mpn.s400      ; }
    ldd hpn    { minor  10; xdd hpn.1         ; }
    rootdev is ldd root_e;
    swapdev is ldd swap;
    dmpdev  is xdd mpn.dump;
}
```

5.3.6.8 Example of the `filesystem` section for a CRAY SSD-T90 device

The following example shows the `filesystem` section of a working CSL parameter file for a CRAY SSD-T90 device.

> **Note:** Additional CSL tags are required in the `unicos` section. See Section 5.3.5, page 49.

```
 filesystem {
        ssdt SSDT00 { iounit 1; tmtype 1; lunit 0;
                iopath { ring 0200; node 01; }
                length 512 Mwords;
                xdd ssd_blk0 { minor  2; block  0; length 131072 blocks; }
                xdd ssd_blk1 { minor  3; block  131072; length 131072 blocks; }
        }
        sdsdev is xdd ssd_blk1;
}
```

5.3.6.9 Example of the `filesystem` section for IOS-E based systems

The following example shows the `filesystem` section of a working CSL parameter file for IOS-E based systems:

```
filesystem {
   disk d0230.0  {type DD62; iopath{cluster 0; eiop 2; channel 030;} unit 0;
       pdd root.0       {minor  10; sector       0; length  166824 sectors;}
       pdd usr.1        {minor  11; sector  166824; length  166824 sectors;}
       pdd core         {minor  12; sector  333648; length  333648 sectors;}
   }
   disk d0232.0  {type DD62; iopath{cluster 0; eiop 2; channel 032;} unit 0;
       pdd root.1       {minor  15; sector       0; length  166824 sectors;}
       pdd usr.0        {minor  16; sector  166824; length  166824 sectors;}
       pdd scratch      {minor  17; sector  333648; length  333648 sectors;}
   }
   disk d0234.0  {type DD62; iopath{cluster 0; eiop 2; channel 034;} unit 0;
       pdd src          {minor 20; sector       0; length  667296 sectors;}
   }
   disk d1230.0  {type DD62; iopath{cluster 1; eiop 2; channel 030;} unit 0;
       pdd mfs0         {minor 60; sector       0; length  166824 sectors;}
       pdd scr1         {minor 61; sector   166824; length  500472 sectors;}
   }
   disk d1232.0  {type DD62; iopath{cluster 1; eiop 2; channel 032;} unit 0;
       pdd mfs1         {minor 62; sector       0; length  166824 sectors;}
       pdd scr2         {minor 63; sector   166824; length  333648 sectors;}
       pdd dump         {minor 64; sector   500472; length  166824 sectors;}
   }
   disk d1234.0  {type DD62; iopath{cluster 1; eiop 2; channel 034;} unit 0;
       pdd stripe0      {minor 70; sector        0; length  667296 sectors;}
  }
   disk d1236.0  {type DD62; iopath{cluster 1; eiop 2; channel 036;} unit 0;
       pdd stripe1      {minor 71; sector        0; length  667296 sectors;}
   }
   sdd strfs {minor 1; pdd stripe0;
                       pdd stripe1;}
   mdd mfs   {minor 1; pdd mfs0;
                       pdd mfs1;}
   ldd root0        { minor  10; pdd root.0      ;}
   ldd usr          { minor  11; pdd usr.0       ;}
   ldd src          { minor  12; pdd src         ;}
   ldd core         { minor  13; pdd core        ;}
   ldd dump         { minor  14; pdd dump        ;}
   ldd bkroot       { minor  15; pdd root.1      ;}
   ldd bkusr        { minor  16; pdd usr.1       ;}
   ldd tmp          { minor  21; pdd tmp0        ;
                                 pdd tmp1        ;}
   ldd usrtmp       { minor  27; pdd usrtmp      ;}
   ldd scratch      { minor  35; pdd scratch     ;
                                 pdd scr1        ;
                                 pdd scr2        ;}
   ldd mir.fs       { minor  30; mdd mfs         ;}
   ldd str.fs       { minor  40; sdd strfs       ;}
   rootdev is ldd root0;
   swapdev is ldd swap;
   dmpdev  is ldd dump;
}
```

5.3.6.10 Example of the `filesystem` section containing an SSD for IOS-E based systems

The following example shows the `filesystem` section of a working CSL parameter file containing an SSD:

> **Note:** Additional CSL tags are required in the `unicos` section. See Section 5.3.5, page 49.

```
filesystem {
   SSD ssddev    {length 512 Mwords;
       pdd ssd_0a       {minor   2; block        0; length   524288 blocks;}
       pdd ssd_0b       {minor   3; block   524288; length   524288 blocks;}
   }
   ldd swap         { minor  23; pdd ssd_0a      ;}
   swapdev is ldd swap;
   sdsdev  is pdd ssd_0b;
   }
```

5.3.7 `network` section

The `network` section defines network devices and network parameters. You can configure them by using the following menu:

```
Configure System
    ->UNICOS Kernel Configuration
        ->Communication Channel Configuration
```

The `network` section includes the following information:

- Descriptions of network parameters
- Customized network device prototypes (IOS-E based systems only)
- Descriptions of each specific network device using standard templates or customized prototypes

The `network` section is specified in the CSL parameter file in the following manner:

```
network {
        number network_parameter_statement;
        custom_network_device_specification_statement;
        physical_network_device_statement;
  }
```

The three statements are repeated as necessary to describe the network device configuration.

The following sections describe the three statement types that compose the `network` section.

5.3.7.1 Network parameters

You can set the network parameters by using the following menu selections:

```
Configure System
    ->UNICOS Kernel Configuration
        ->Network Parameters
```

and

```
Configure System
    ->Network Configuration
        ->TCP/IP Configuration
            ->TCP Kernel Parameters Configuration
```

The network parameter statement has the following format:

```
number network_parameter_statement;
```

The *number* is a valid CSL number for the network statement. *network_parameter_statement* argument can have the values described in Table 31 through Table 35, page 77.

Table 31. Network parameter values (common)

Parameters	Description
`atmarp_entries`	Size of the asynchronous transfer mode (ATM) address resolution protocol (ARP) table.
`atmarp_recv`	Amount of socket space used for `receive` for ATM ARP traffic. Should be a power of 2.
`atmarp_send`	Amount of socket space used for `send` for ATM ARP traffic. Should be a power of 2.

Parameters	Description
`cnfs_static_clients`	Maximum number of active Cray NFS static clients.
`cnfs_temp_clients`	Maximum number of active Cray NFS temporary clients.
`hidirmode`	Sets permissions or file mode for the following directories: `/dev/ghippi#` (GigaRing based system) and `/dev/hippi` (IOS-E based system).
`hifilemode`	Sets permissions or file mode for the following files: `/dev/ghippi#/*` (GigaRing based system) and `/dev/hippi/*` (IOS-E based system).
`nfs_duptimeout`	Time interval in seconds during which duplicate requests received by the NFS server will be dropped.
`nfs_maxdata`	Maximum amount of data that can be transferred in an NFS request (the NFS data buffer size).
`nfs_maxdupreqs`	Size of the NFS server's duplicate request cache.
`nfs_num_rnodes`	Size of the NFS client's NFS file-system-dependent node table (rnode table for NFS).
`nfs_printinter`	Time in seconds between `server not responding` message to a down server.
`nfs_static_clients`	Number of static client handles reserved for NFS client activity.
`nfs_temp_clients`	Number of dynamically allocated client handles that can be used for NFS client activity when all of the static client handles are in use.
`nfs3_async_max`	Maximum amount of data (in 4096–byte blocks) that will be written per file asynchronously. After this value is exceeded, all writes will be synchronous. The default is 2000 (2000 * 4096 = 8,192,000 bytes).
`nfs3_async_time`	The amount of time (in seconds) that data will be held in the NFS async write cache on the client.
`tcp_numbspace`	Number of clicks of memory set aside for TCP/IP managed memory buffers (MBUFs).

Table 32. Network parameter values (GigaRing based systems)

Parameters	Description
`maxinputs`	Maximum number of asynchronous read I/O requests that can be issued to the I/O node. This must be an integer value in the range 1 through 256. The default is 16.
`maxoutputs`	Maximum number of write I/O requests that can be issued to the ION. This must be an integer value in the range 1 through 256. The default is 16.
`maxusers`	Maximum number of applications that can share the HIPPI device. This parameter applies only to HIPPI devices; it must be set to 1 for other interfaces. For HIPPI devices, the value must be an integer in the range 1 through 8. The default is 2.

Table 33. Network parameter values (IOS-E based systems)

Parameters	Description
`atmmaxdevs`	Maximum number of ATM devices.
`fdmaxdevs`	Maximum number of fiber distributed data interface (FDDI) FCA-1 channel adapters.
`himaxdevs`	Maximum number of channels that you can configure for HIPPI.
`himaxpaths`	Maximum numbers of logical paths per channel that you can configure for HIPPI.
`npfilemode`	Sets the permissions for `/dev/comm/`*CHAN*`/lp`*XX*files.
`npmaxdevs`	Maximum number of channels that can be configured for low-speed channel communication devices.
`npmaxpaths`	Total number of logical paths for low-speed channel communication devices.

Parameters	Description
`npmaxppd`	Maximum number of paths per device for low-speed channel communications devices.
`npoto`	Output time-out.
`nprthresh`	Reads threshold for low-speed channels (number of reads that can be queued to the IOS at one time).
`nprto`	Default read time-out (per path).
`npwthresh`	Writes threshold for low-speed channels (number of writes that can be queued to the IOS at one time).

Table 34. Network parameter values (IOS-E based systems: CRAY J90 only)

Parameters	Description
`enmaxdevs`	Maximum number of Ethernet devices.

Table 35. Network parameter values (IOS-E based systems: CRAY T90 and CRAY C90 only)

Parameters	Description
`fddirmode`	Sets permissions of file mode for `/dev/fddi*` directories.
`fdfilemode`	Sets permission of file mode for `/dev/fddi*/*` files.
`fdmaxpaths`	Maximum number of logical paths per physical FDDI channel.
`npdirmode`	Sets the permissions for subdirectories of `/dev/comm`.
`npito`	Input time-out.

For more information about ATM, see the *Asynchronous Transfer Mode (ATM) Administrator's Guide*, Cray Research publication SG–2193.

5.3.7.2 Customized network device prototypes for IOS-E based systems

The customized network device prototype lets you define characteristics for low-speed devices unique to your site. The network device prototype statement has the following syntax:

```
np_spec interface_type {
        { device type device_type;
        channel mode channel_mode;
        driver type driver_type;
        driver mode driver_mode_number;
        device function function_number;
  direction timeout timeout_number;
       }
```

interface_type	Type of the interface as defined in Table 36, page 80.
device_type	Type of the device as defined in Section 5.3.7.3.

channel_mode	One of the following values:
	`12MB`
	`12MB_LP`
	`6MB`

driver_type	One of the following values:
	`FY`
	`LCP`
	`MP`
	`NSC_MP`
	`PB`
	`RAW`

driver_mode_number	Number of the driver mode.
function_number	Number of the function.

direction	Either `input` or `output` (only used with HIPPI devices).
timeout_number	Time-out value.

5.3.7.3 `device type`

The `device type` statement, which defines the network device type, has the following format:

```
device type device;
```

The following are valid device types:

Device type	Description
`A130`	NSC A130 devices
`CNT`	CNT devices
`DIAG`	Diagnostic devices
`DX4130`	FDDI connection
`DX8130`	FDDI connection
`EN643`	IP router
`FEI_3`	FEI-3 devices
`FEI_3FY`	FEI-3FY devices
`FEI_4`	FEI-4 devices
`FEI_CN`	FEI-CN devices
`FEI_DS`	FEI-1 data-streaming IBM connection
`FEI_UC`	FEI-1 user driver channel
`FEI_VA`	VAX FEI-1, port A
`FEI_VB`	VAX FEI-1, port B
`FEI_VM`	Cray devices
`N130`	NSC N130 devices
`VME`	VME-bus devices

Cray Research provides you with configuration templates for a number of standard interfaces, as shown in Table 36.

Table 36. Standard interface configuration templates

Interface type	Device type	Channel mode	Driver type	Driver mode	Device func	Input time-out	Output time-out	Read time-out
S_DIAG6	DIAG	6MB	RAW	0	0	100	100	100
S_DIAG12	DIAG	12MB	RAW	0	0	100	100	100
S_DIAG12L	DIAG	12MB_LP	RAW	0	0	100	100	100
S_FEICN	FEI_CN	6MB	LCP	0	0	100	100	600
S_FEIDS	FEI_DS	6MB	LCP	0	0	100	100	600
S_FEIUC	FEI_UC	6MB	LCP	0	0	100	100	600
S_FEIVA	FEI_VA	6MB	LCP	1	0	100	100	600
S_FEIVB	FEI_VB	6MB	LCP	2	0	100	100	600
S_FEIVM	FEI_VM	6MB	LCP	0	0	100	100	600
S_FEI3	FEI_3	6MB	MP	0	0	100	100	600
S_FEI312	FEI_3	12MB	MP	0	0	100	100	600
S_VAXBI	VAX_BI	12MB	MP	1	0	100	100	600
S_N130X	N130	12MB	PB	0	0	100	100	600
S_EN643X	EN643	12MB	PB	0	0	100	100	600
S_DX4130X	DX4130	12MB	PB	0	0	100	100	600
S_DX8130X	DX8130	12MB	PB	0	0	100	100	600
S_FEI3FY	FEI_3FY	6MB	FY	010	0	100	100	600

Interface type	Device type	Channel mode	Driver type	Driver mode	Device func	Input time-out	Output time-out	Read time-out
`S_FEI4`	`FEI_4`	`6MB`	`FY`	`010`	`0`	`100`	`100`	`600`
`S_A130X`	`A130`	`12MB`	`NSC_MP`	`0`	`0`	`100`	`100`	`600`

If your site has a network device that does not match these definitions, you must customize a device definition by using the customized network device prototypes, as described in this section. The menu that you would use to do so is as follows:

```
Communication Channel Configuration
    ->Custom network device specification
```

5.3.7.4 Network devices

The network device statement, which describes specific devices, consists of the following statements:

- `iopath` (see Section 5.3.6.1, page 55, for syntax information)
- `logical path` or (for high-speed channels) *direction* argument
- `device type`
- `np_spec`
- `fddi value`

More than one I/O path, logical path, device prototype, device type, and direction combination can be specified for a given device.

The `iopath` syntax is discussed in Section 5.3.6.1, page 55.

5.3.7.5 Device types

The following tables describe the types of devices for GigaRing based systems and for IOS-E based systems.

Table 37. Network device types (GigaRing based systems only)

Device type	Description
`gfddi`	GigaRing FDDI device.
`gatm`	GigaRing asynchronous transfer mode (ATM) device.
`gether`	GigaRing Ethernet device.
`ghippi`	GigaRing HIPPI device.

Table 38. Network device type (IOS-E based systems only)

Device type	Description
`fddev`	IOS-E FDDI device.
`hi`	High-speed HIPPI device.
`npdev`	Low-speed channel; if this device is specified, then the *number* argument is the ordinal of the network device.

5.3.7.6 Device formats

The following sections describe device formats for GigaRing based systems and for IOS-E based systems.

5.3.7.6.1 Device formats for GigaRing based systems

The following formats apply to GigaRing based systems:

```
gatm 0 {
                  iopath {
                           iopath_information
                  }
                  maxusers number;
                  maxinputs number;
                  maxoutputs number;
          }
```

```
gether 0 {
                   iopath {
                            iopath_information
                   }
                   maxusers number;
                   maxinputs number;
                   maxoutputs number;
           }
```

```
gfddi 0 {
                   iopath {
                            iopath_information
                   }
                   maxusers number;
                   maxinputs number;
                   maxoutputs number;
           }
```

```
ghippi 0 {
                   iopath {
                            iopath_information
                   }
                   maxusers number;
                   maxinputs number;
                   maxoutputs number;
           }
```

5.3.7.6.2 Device format for IOS-E based systems

The following formats apply to IOS-E based systems only:

```
fddev number {
        padcnt number;
        treq number;
        maxwrt number;
        maxrd number;
        iopath {
             iopath_information
       }
  }
```

```
hidev number {
          iopath {
              iopath_information
          }
          logical path number {
                  flags number;
                  I_field number;
                  ULP_id number;
          }
          flags number;
          input;
          device type type;
  }
```

```
npdev number {
       iopath {
            iopath_information
       }
       np_spec name;
  }
```

5.3.7.7 `logical path` for IOS-E based systems

The `logical path` statement has the following format:

```
logical path number {
       identifier number
  }
```

If you have a high-speed HIPPI network device, you could use the *identifier* argument, which has the following values:

Identifier	Description
`UL_id`	Upper-level protocol identifier; it is a value between 0 and 255 that identifies the protocol or user of a HIPPI packet. The value is the first byte of each HIPPI packet and is used with `input` to direct incoming packets to the appropriate process (TCP/IP uses a value of 004).

`I_field`	A data value that is passed over the HIPPI channel when a connection is made. In `HXCF_DISC` mode, the `I-field` value precedes each packet. The `I-field` value selects the output port of an NSC P_8 or PS_32 switch.
`flags`	Flags related to the specific HIPPI logical device. The only flag available at this time is `02000` (`HXCF_IND`). When this flag is set, the driver interprets the first word of each user output buffer as the `I-field` value. The driver puts the incoming `I-field` value in the first word of the user input buffer.

5.3.7.8 *direction* argument for IOS-E based systems

The *direction* argument has the following syntax:

```
direction;
```

The *direction* argument (used only with HIPPI devices) can be either `input` or `output`.

5.3.7.9 `device type` statement for IOS-E based systems

The `device type` statement has the following syntax:

```
device type device ;
```

device defines the device connected to the HIPPI channel.

5.3.7.10 `np_spec` statement for IOS-E based systems

The `np_spec` statement has the following syntax:

```
np_spec name
```

name specifies the customized network device specification describing the network device connected to the low-speed channel.

5.3.7.11 `network` section example common to GigaRing based and IOS-E based systems

The following example configures FDDI, HIPPI, and NP devices and only those custom network device specifications (`np_spec`) that are needed for an IOS-E based system:

```
network {
    8 nfs_static_clients;
    8 nfs_temp_clients;
    8 cnfs_static_clients;
    8 cnfs_temp_clients;
32768 nfs_maxdata;
  256 nfs_num_rnodes;
 1200 nfs_maxdupreqs;
    3 nfs_duptimeout;
    0 nfs_printinter;
 8000 tcp_nmbspace;
 0700 hidirmode;
 0600 hifilemode;
```

5.3.7.12 `network` section example for GigaRing based systems

The following is an example of a `network` section for a GigaRing based system:

```
network {
    gether 0 {
        iopath { ring 01; node 02; channel 05; }
        maxusers 1;
        maxinputs 64;
        maxoutputs 64;
    }
    gfddi 0 {
        iopath { ring 01; node 02; channel 04; }
        maxusers 1;
        maxinputs 64;
        maxoutputs 64;
    }
    gatm 0 {
        iopath { ring 01; node 03; channel 01; }
        maxusers 1;
        maxinputs 64;
        maxoutputs 64;
    }
    ghippi 0 {
        iopath { ring 01; node 04; channel 02; }
        maxusers 4;
        maxinputs 64;
        maxoutputs 64;
    }
}
```

5.3.7.13 `network` section example for IOS-E based systems

The following example configures FDDI, HIPPI, and NP devices and only those custom network device specifications (`np_spec`) that are needed:

```
network {
    4 npmaxdevs;
   32 npmaxpaths;
   16 npmaxppd;
  100 npito;
  100 npoto;
  600 nprto;
   10 nprthresh;
   10 npwthresh;
    4 himaxdevs;
    8 himaxpaths;
    1 fdmaxdevs;
   16 fdmaxpaths;
0700 fddirmode;
0600 fdfilemode;
0700 npdirmode;
0600 npfilemode;
np_spec FEI3FY {
        device type FEI_3FY;
        channel mode 6MB;
        driver type FY;
        driver mode 8;
        device function 0;
        input timeout 100;
        output timeout 100;
        read timeout 600;
}
np_spec N130X {
        device type N130;
        channel mode 12MB;
        driver type PB;
        driver mode 0;
        device function 0;
        input timeout 100;
        output timeout 100;
        read timeout 600;
}
npdev   0 {
        iopath { cluster 0; eiop 0; channel 030; }
        np_spec FEI3FY;
}
npdev   1 {
        iopath { cluster 0; eiop 0; channel 032; }
        np_spec N130X;
}
hidev   0 {
        iopath { cluster 0; eiop 3; channel 030; }
        logical path 0 {
                flags 00;
                I_field 00;
                ULP_id 00;
        }
        flags 00;
        input;
        device type PS_32;
}
hidev   1 {
        iopath { cluster 0; eiop 3; channel 032; }
        logical path 0 {
                flags 00;
                I_field 00;
                ULP_id 00;
        }
        flags 00;
        output;
        device type PS_32;
}
fddev   0 {
        treq 8;
        padcnt 3;
        maxwrt 10;
        maxrd 10;
        iopath { cluster 0; eiop 0; channel 034; }
        logical path 0 {
                rft ALL;
                read timeout 60;
        }
}
}
```

5.3.8 `revision` section

The `revision` section marks the CSL parameter file with a site-defined name for identification purposes, particularly for programs and other Cray Research products. The `revision` string is set automatically when you use the ICMS.

The `revision` section is specified in the CSL parameter file by the following statement:

`revision` *text_string*

The *text_string* should be a string that is significant for your site and allows you to identify the file.