# Helios for the Sun Workstation

# Copyright





POSIX refers to the IEEE standard 1003.1-1988, *Portable Operating System Interface for Computer Environments*.

This document was printed in the U.K.

November 1989

Part number: DM5021

# Preface

This guide explains how to install and then run Helios on your workstation. It supplements the information given in the Helios V1.0 manual, *The Helios Operating System* (Prentice Hall, 1989). The guide is divided into three chapters: the first chapter describes the actual installation procedure, the second chapter describes the I/O Server, and the third outlines the additions and changes present in Helios V1.1. The information in Chapter 3 also appears in the *Helios Network Guide* (DSL, 1989). Before starting work on the installation, we suggest that you read the first two chapters carefully.

Although there are occasional examples that refer to specific hardware, this document is intended to act as a generic guide to installing Helios on a Sun, or an equivalent host, with additional transputer board(s). At present, you can run Helios on a Sun3 or Sun4 Workstation with the following transputer boards:

CSA PART.8
Inmos B011
Inmos B014
Inmos B016
Niche NT1000
Parsytec/Paracom Sun Board
Transtech Multi-Computing Platform (MCP1000)

You can also run Helios on a Telmat UNIX workstation host with an ITFTP32 transputer board. This machine is roughly equivalent to a Sun Workstation and so you can equally well apply the information provided here to install Helios on your Telmat hardware.

The list of supported hardware is growing all the time. To find out what is currently supported, contact your local Helios dealer, or write directly to Distributed Software Limited, 1900 Aztec West, Almondsbury, BS12 4SD.

# Table of Contents


**Chapter 1 Installing Helios on Sun Workstations**

**1.1 Introduction**

**1.2 Installation**

**1.3 Startup Options**

**Chapter 2 The Helios Unix I/O Server**

**2.1 The Windowing Interface**

2.1.1 Sunview

2.1.2 Dumb Terminals

**2.2 The Filing System Interface**

**2.3 The Error Logger**

**2.4 Multiple Links**

**2.5 Hydra**

**2.6 Other host.con Configuration Entries**

**Chapter 3 Helios Version 1.1**

**3.1 Job Control in the Shell**

**3.2 Alias Server**

**3.3 Batch Server**

**3.4 Fault Library**

3.4.1 Fault

3.4.2 Low Level Routines

3.4.3 Fault Database Format

**3.5 Technical Changes**

**3.6 Component Distribution Language**

3.6.1 Component Attribute Code

3.6.2 Subscripted Component Declarations

3.6.3 Replicators

3.6.4 Multi-Dimensioned Replication

3.6.5 Iteration Names

3.6.6 Precedence of Constructors

3.6.7 Automatic Allocation of Streams

3.6.8 Subscript Expressions

3.6.9 CDL Compiler

3.6.10 CDL Scripts

3.6.11 New CDL Syntax

**3.7 New Commands in Helios 1.1**

**3.8 Network Support**
3.8.1 Control System
3.8.2 Network Commands
**3.9 The Session Manager**
**3.10 New Functions**
**3.11 Changed Functions**
**3.12 Extended Functions**

# Chapter 1

# Installing Helios on Sun Workstations

This chapter describes the Helios/Sun release and explains how you can install it on your hardware, which must be using SunOS 4.01 or a later version. The first part of this chapter outlines the installation procedure and then describes the work involved in installing Helios for a number of different configurations. The second part of this chapter specifies the various initialisation options that you can use to define the way in which Helios starts up.

## 1.1 Introduction

The basic unit of hardware used by the Sun I/O Server is the **transputer site.** A site consists of a link adapter attached to at least one transputer. In the current implementation, every active user of Helios needs his or her own site: this site may be accessed directly, with the I/O Server running on the Sun with the transputer hardware; or it may be accessed indirectly, running the I/O Server on a remote Sun and a link daemon, **Hydra**, on the Sun with the transputer hardware to provide access to the link. Sites are given numbers from 0 onwards. For example, the CSA PART.8 board has four link adapters and hence four sites, site 0 - site 3.

The Helios/Sun product comes on a single cartridge tape, written to */dev/rst8* on a Sun-3 unless otherwise requested; no special blocking factors are used. The tape contains two directories: *sunbin*, which has a number of executables to run on the Sun host; and *helios*, which contains the normal Helios files for the transputer side.

The directory structure for *helios* is as follows:

| Directory Name | Contents |
|---|---|
| *bin* | Various user commands |
| *lib* | Binary files used by the system |
| *include* | The C compiler's include files |
| *tmp* | Temporary files |
| *etc[0...n]* | Various system textual resource files, some of which depend upon the hardware attached to the different sites |
| *etc.master* | A backup copy of the various *etc* directories |

## 1.2 Installation

The following steps outline the different stages in the intallation process.

1. The first stage in the installation process is to extract the *sunbin* and *helios* directories from tape, using

   ```
   tar fxv /dev/rst8
   ```

   Note that you will require approximately two megabytes of disc space. After extracting the directories it may be necessary to modify the owner, group, and access permission of the files to reflect the conventions at your installation (an **installation** can be a Sun host and hardware or an entire network of Suns plus additional transputer hardware). In most circumstances, users need read access to *helios/bin*, *helios/lib*, and *helios/include*, but read/write access to *tmp* and the various *etc* directories.

2. The next stage is to install the various Sun executables in the *sunbin* directory, so that they can be accessed by all users. The executables exist in Sun-3 and Sun-4 format. In theory the Hydra link daemon and the hydramon monitor program do not have to be available to ordinary users, only to the installation administrator, but it may be easier during the initial stages to allow general access to these programs. The *serverwindow* and *serverwindow.sun3* programs are run by the Server as separate processes, never by the ordinary user, and should be kept in the *helios* directory; *serverwindow* is the Sun-4 version, and *serverwindow.sun3* is for the Sun-3. The I/O Server must always be accessible to ordinary users.

3. As the default Helios directory, */home/sp2sun1/user1/helios*, is unlikely to be correct for your installation. You must next modify the master *host.con* Server configuration file, to reflect the location of the Helios directory within your host's filing system. After you have updated the master copy, every user should make his own copy of the configuration file in his own home directory, and always run the I/O Server from that directory, never from within the *helios* directory. A user can make his own copy of the *helios* directory and work within that, modifying his private *host.con* file, but this will use an additional 2 Mbytes of disc space for each user. Notice that the Server will always pick up the *host.con* file from the current directory, unless you introduce an alternative filename with the -C option when you invoke the **server** command.

4. The final part of the installation process determines the networking and transputer site allocation, which depends very much on both the hardware and the users' requirements. The available options are described here, along with an outline of the work involved, but this description should only be regarded as a guide line.

### 1.2.1 Option 1

The simplest option involves a single Sun (not networked) with a single transputer plug-in board containing four transputers and four link adapters (for example, a Transtech MCP1000 board). This configuration is intended for four users, each with just one transputer. In this case the Server will always run on the Sun with the board, so there is no need to go via the link daemon, Hydra. All the sites are equivalent and there is no need for users to be allocated a particular site because of the hardware attached to that site. Every user should therefore comment out the line

*site = x*

in their configuration file, *host.con*, which means that they will be allocated any free site when they run the I/O Server. The installation administrator should also comment-out the lines

```
run -r startns
```

and

```
waitfor /tfm
```

from */helios/etc[0...n]/initrc*, as there is no need to run the Network Server on a single transputer network.

## 1.2.2 Option 2

The next level of complexity is to have the same hardware as described in Option1, but this time with a maximum of two users, each with access to two processors. On a Transtech MCP1000 board this means that site 0 uses the processors corresponding to */dev/nap0* and */dev/nap1*, and site 2 uses the processors corresponding to */dev/nap2* and */dev/nap3*. Under no circumstances should the Server try to use sites 1 or 3, as there is no way to detect from the Sun side that the corresponding root processors are already being used. Before running the Server, connect the processors together using the **nt_ctl** utility. Each user's configuration file, *host.con*, should contain a reference to either site 0 or site 2, but note that the Server will issue an error message if that site is already being used. As each user now has a network of two transputers, you should use the Network Server to boot-up the second processor; */helios/etc[0...n]/initrc* should not be modified. You should also ensure that the resource map specifies the correct reset driver, *tram_ra.d*.

You can build networks with three or four transputers, using just the transputer board, by adjusting the network resource maps and the *host.con* file as appropriate.

## 1.2.3 Option 3

The next level of complexity is to attach additional transputers to some or all of the sites, but keeping the resulting networks separate. This would allow four users, each with multiple transputers. If all four sites have the same network attached there is no reason to run on any particular site, so the *host.con* files should not specify a particular site. If the sites have different configurations, users may wish to run on a particular site, and they can specify this in *host.con*. Once this level of complexity is reached, it is assumed that the various users of the system will cooperate with each other.

If all four sites have different network configurations they will require different network resource maps, which is fairly straightforward as every site has its own private copy of */helios/etc[0...n]*: *etc0* is used for site 0, and so on. The reset driver used for the network depends on the hardware.

Note that any sites not currently used by Helios can be used for other applications, such as TDS.

### 1.2.4 Option 4

The final option allows all the sites to be accessed remotely over the ethernet. In this case it is necessary to run the link daemon, Hydra, on a Sun workstation with a transputer board, and this requires a number of installation steps.

1. Add the Hydra internet service to the system configuration file, */etc/services*, as a tcp service, with a unique socket number; this addition must be made on every machine that will run either Hydra or the Server.

2. Modify the *hydra.con* configuration file. In particular, you must change the line specifying the hydra_host to indicate the network address of the machine running Hydra; this network address must correspond to an entry in the */etc/hosts* file.

3. Modify the *host.con* file to indicate a remote transputer box; again, the hydra_host line must be modified.

If you wish, you can allow remote access to only some of the sites by listing those sites rather than all_sites in the *hydra.con* configuration file. The remaining sites can then be accessed directly, which is more efficient than going via Hydra, but does not allow networked access. You can also change which sites are accessible remotely using the hydramon program. This is discussed in more detail in the next chapter.

## 1.3 Startup Options

When Helios is booted into a transputer that is connected to the I/O processor, it executes */helios/lib/init*, which in turn reads the text file *etc[0...n]/initrc*. This text file can be edited by the user to define the way in which Helios is to start, and can contain any of the commands listed below.

*# comment*

Any line in *initrc* which starts with # is treated as a comment and is ignored.

**auto** *servername*

The **auto** command adds the specified name, *servername*, into the Name Server. This means that the server will be loaded on demand from /helios/lib when it is first used.

**console** *servername windowname ...*

The **console** command creates one or more windows using the specified server, *servername*, and provides the window *windowname* as output for commands executed from within initrc. Until you specify **console**, any output from tasks such as the Network Server will go to the */logger* device. Note that in the I/O Server the standard *initrc* file contains an entry at the end to run login; this will only work if a **console** command has been given in order to create a window. If no console has been created then the login process must be created by the Batch Server by adding a suitable entry in the file *etc/batchrc*, which provides an environment for **login**.

**ifabsent** *servername command*

The **ifabsent** command determines if the specified server, *servername*, exists; if that server does not exist, it treats the rest of the line as a command which it will execute. *command* must be one of the commands listed in this section.

**run** *[-e] [-w] pathname [ argv ... ]*

**run** executes the command whose full pathname is passed as the argument *pathname*. The **-e** option passes the environment to the command, and the **-w** option forces *run* to wait for the command to terminate before it terminates itself. The following argument vector, *argv*, consists of the command name and each of its arguments; it is only used if you specify the **-e** option. Any program started using **run -e** will

subsequently have standard streams to a window created by **console**, or to the error logger if no console has been specified.

**waitfor** *server_name*

The **waitfor** command waits for the server specified as *server_name* to be loaded before it terminates. If the server has not been loaded, **waitfor** will result in it being loaded (assuming that the name is defined).

The following is a listing of the configuration file that is shipped with this release of Helios.

```
# Helios System Configuration File
# This file is interpreted by init to configure the system
# it is NOT a shell script.
ifabsent /window auto /window
waitfor /window
console /window console
run -e /helios/bin/startns startns -r /helios/etc/default.map
waitfor /tfm
#run -e /helios/lib/sm sm
run -e /helios/lib/bs bs
#run -e /helios/bin/smlogin -
run -e /helios/bin/login -
```

If the Window Manager is running as part of the I/O Server, the test

```
if absent /window
```

will fail and the system will create a console window using the host's windowing system. If the Window Manager is not part of the I/O Server then the first two command lines in this program will install the Window Manager: **auto** adds the Window Server's name into the Name Server, **waitfor** then waits for the server to be loaded. This may appear to be a somewhat verbose way of loading the Window Server, but it is necessary because **run** alone would only load the server; it would not create an entry within the Name Server. It should be noted, however, that some servers add their own names into the Name Server, and can therefore be loaded successfully with **run**.

Having created the Window Server, the program uses the **console** command to create a window called *console*. The environment provided by this window is then noted by **init**, so that it can be passed on to subsequent **run** commands that are invoked with the **-e** option. In this configuration file, all subsequent **run** commands are passed this environment so that their output is sent to the console window instead of to */logger*.

After applying the **run** command to **startns**, the program loads the Batch Server. One of the first tasks that the batch server performs is to read the job description file, *etc/batchrc*. In this release of Helios the job description file is empty, but it can be modified by the user. Full details of the syntax are given in the description of the Batch Server in Section 3.3.

# Chapter 2

# The Helios Unix I/O Server

This chapter describes the various facilities provided by the Helios I/O Server, version 3.72, when running on a Unix machine. It is intended to supplement the chapter on the I/O Server in the Helios manual, *The Helios Operating System* (Prentice Hall, 1989), and is aimed primarily at the Sun I/O Server with a transputer board (although it should also be relevant to other Unix versions). In particular, this chapter describes the various ways of configuring the I/O Server using the *host.con* configuration file. This configuration file usually resides in the current directory, and is used when the Server starts up. It is possible to specify a different configuration file on the command line using the -C option, for example,

```
server -C ../../dumbterm.con
```

This option is particularly useful when combined with shell aliases, allowing the user to have different commands for the different configurations. Every user should have his or her own copy or copies of this configuration file.

## 2.1 The Windowing Interface

The Sun I/O Server can provide multiple windows on the host side, either using real windows on a SunView display or multiple pseudo-windows with hot-key switching on a dumb terminal; the latter gives a similar environment to the I/O Server used with Helios/PC. To determine which windowing system to use the I/O Server examines the TERM environment variable: if this is set to "sun", real windows will be used; pseudo-windows are used in all other cases.

It is possible to disable multiple windows on the host side and use multiple windows on the transputer side instead, by commenting out the line Server_windows in the *host.con* file. In this case the transputer will run the program */helios/lib/window*, which should provide multiple windows one way or another. This option is unlikely to be useful for the Sun I/O Server.

### 2.1.1 SunView

When running the I/O Server under SunView, Helios will use real SunView windows for its own windowing operations. All Helios operations which create a window will therefore create a new SunView window. These windows use the standard Helios escape sequences for input and output, as documented in *The Helios Operating System*. In addition, the I/O Server will inherit the special tty keys for *erase*, *intr*, *start*, and *stop*, and will map these onto the Helios equivalents.

Note that most programs only check the screen size at start-up, and cannot therefore be correctly used with windows that are resized during program execution. This is a result of the way in which each individual application has been written, and is not a restriction that is imposed by the I/O Server.

The I/O Server has its own window for its debugging output. This window has a control panel for the various facilities available within the I/O Server. There are buttons for rebooting Helios, for terminating the I/O Server and returning to Unix, for entering the I/O Server's low-level debugger, and for obtaining the Server status. The error logger destination can be toggled, and there is a pop-up menu for the debugging facilities. The left mouse button can be used to enable or disable all debugging options, and the right button can be used to select a particular option as follows (the keys in brackets are used with dumb terminals):

| Option | Action |
|---|---|
| Resources (x) | List all open streams. |
| Reconfigure (z) | Re-read the *host.con* configuration file. Please note that some of the options do not take effect until you reboot the transputer, and some of the options are only checked when the Server starts up. |
| Messages (m) | Report on messages sent by/to the transputer network. |
| Search (s) | Report all distributed searches. |
| Open (o) | List all files being opened. |
| Close (p) | List all files being closed. |
| Name (n) | Give the names of all objects Helios tries to access. |
| Read (r) | Report all file reads. |
| Boot (b) | Progress report during transputer bootstrap. |
| Keyboard (k) | Report all key presses. |
| Init (i) | Progress report while device servers are starting. |
| Write (w) | Report all file writes. |
| Quit (q) | Give a progress report while the Server is exiting. |
| Graphics (g) | Report any graphics operations. |

These debugging options can also be enabled on the command line. For example, **server -opr** starts the I/O Server with the *open*, *close*, and *read* debugging options enabled. This is compatible with the PC version of Helios.

For a number of reasons the I/O Server has to fork a new program whenever it creates a new window. Usually this is **serverwindow** for a Sun-4, or **serverwindow.sun3** for a Sun-3, but it is possible to specify some other

program in the configuration file. The following example would cause the Server to execute **myservwindow**:

```
serverwindow = myservwindow
```

### 2.1.2 Dumb Terminals

When the I/O Server is executed from a dumb terminal, the Server will use its own windowing system. In this environment only one window is visible at a time, although the others can be viewed by applying special key sequences. Output to non-visible windows proceeds normally, and will become visible when the user switches to that window. The termcap database and the TERM environment variable are used by the I/O Server to interpret the standard Helios output escape sequences, and to map the terminal's input to the Helios input sequences when necessary.

The I/O Server has its own window that is not directly accessible from Helios. This window is used for the Server's error messages, and may be used as the destination for the error logger. When output is written to the Server's window, this window will pop to the foreground allowing the error messages to be observed. To disable this option, you can insert the line:

```
Server_windows_nopop
```

in the configuration file.

The I/O Server uses a number of special keys or key sequences to control reboots, debugging options, window switching, and so on. The keys for these operations may have to be different for different terminals, so they can be defined by the user by adding entries into the configuration file. A possible *host.con* entry is

```
escape_sequence = k1
```

This specifies that the main hot-key is function key 1, k1 being the **termcap** name for that function key. Other termcap names are described in the standard Unix documentation, but the most common ones are k1-k9, which represent the first ten function keys, kh for the home key, and ku, kd, kr, and k1, for up-arrow, down-arrow, right-arrow, and left-arrow respectively.

To perform the various Server operations, you simply press the appropriate hot-key, followed by another key, as specified below.

| Keys | Operation |
|---|---|
| <*hot key*> 1 | Switch to next window. |
| <*hot key*> 2 | Switch to previous window. |
| <*hot key*> 3 | Refresh current window. |
| <*hot key*> 7 | Enter debugger. |
| <*hot key*> 8 | Server status. |
| <*hot key*> 9 | Server exit. |
| <*hot key*> 0 | Reboot transputer. |
| <*hot key*> a | Toggle all debugging options. |
| <*hot key*> l | Switch error logger destination. |
| <*hot key*> x | Resource debugging. |
| <*hot key*> z | Re-read configuration file. |
| ... | (More operations may be added) |

The same mechanism can be applied to the debugging options described earlier.

Some of these operations are used more often than others, and it is convenient to have them as single-key operations rather than a two-key sequence. The following lines can be added to the *host.con* file to assign these operations to function keys:

```
switch_forwards_key  = k2
switch_backwards_key = k3
refresh_key          = k4
debugger_key         = k5
status_key           = k6
exit_key             = k7
reboot_key           = k8
```

Your terminal may have keys for which there is no termcap entry, or for which the termcap entry is incorrect. You can still use these keys as escape keys if you specify the key's data in the *host.con* file. If you need do this, you must always prefix the key's data with a '#' character. For example,

```
escape_sequence = #\E^Q\0120\n
```

specifies that the main hot key generates an escape character (hex 0x1B), followed by a CTRL-Q (hex 0x11), the octal number 12 (hex 0x0A, or ASCII linefeed), the letter 'O' (hex 0x4F), and another linefeed (hex 0x0A). You can enter a space by using its octal value, 040, and backslash and caret by using \\ and \^ respectively.

The I/O Server translates the termcap sequences into the following Helios sequences:

| **Termcap** | **Helios** |
|---|---|
| k1-k9 | Function keys 1 to 9 |
| k; | Function key 10 |
| &8 | Undo |
| @7 | End |
| kI | Insert |
| kN | PageDown |
| kP | PageUp |
| kh | Home |
| kd | Down-arrow |
| ku | Up-arrow |
| kr | Right-arrow |
| kl | Left-arrow |
| %1 | Help |

If any of these are used as special keys for the I/O Server they cannot be read by Helios; any other keys are passed to Helios without translation. Helios programs should not, in general, make assumptions about the keys that will and will not be available on a particular terminal.

To perform the translation of Helios escape sequences to screen operations, the I/O Server uses the following termcap entries:

| Termcap | Description |
|---|---|
| **bl** | Bell sequence; used in preference to CTRL-G |
| **cl** | Clear screen |
| **cm** | Cursor move |
| **mr** and **me**, or **so** and **se** | Inverse/Normal video |
| **ce** | Clear to end of line |
| **am** | Determine some of the terminal's wrapping characteristics |
| **ro** and **co** | Determine the terminal size |

If any of the above are not defined correctly, then the Server's behaviour is undefined. In addition, the exact nature of a terminal's line wrapping may cause the display to become confused, so a screen refresh key-sequence is provided.

## 2.2 The Filing System Interface

The I/O Server provides two Helios servers to allow access to the host's filing system. The first is */helios*, which contains all the standard Helios files and binaries. The second is */files*, which maps onto the root of the Unix filing system, so that, */Cluster/IO/files/usr/games* is the same as the Unix directory */usr/games*. All files and directories on the Unix filing system are accessible from Helios, including networked drives. However, Helios does not provide its users with any access authority other than their standard Unix ones.

The location of the main Helios directory depends on the site, and must be specified in the *host.con* file, together with some other files, as follows:

```
helios_directory = /home/sp2sun1/helios
bootfile         = ~/lib/nboot.i
system_image     = ~/lib/nucleus
```

The tilde characters (~) in the last two entries indicate that they are relative to the *helios* directory, so in most cases it is only necessary to change the *helios_directory* entry when installing Helios.

The various Helios files in the standard release occupy over a megabyte of disc space, which is compact for a complete operating system, but nevertheless it is not a good idea to have more than one copy of these files in an installation. This creates a number of problems. Firstly, different network maps may have to be used to boot up different transputer sites, even though the Helios initialisation file */helios/etc/initrc* only specifies one, which is usually */helios/etc/default.map*. Also, copies of Helios are serialised and there is checking within Helios to prevent multiple users from using the same copy of Helios. To overcome this, all accesses to the */helios/etc* directory and to the */helios/lib/net_serv* program are modified according to the site used. For example, if the user is connected via site 2 and tries to access */helios/etc/motd*, the Server will actually access *etc2/motd* within the *helios* directory.

Because the entire Unix filing system is accessible from Helios, the user can also access the various devices and other objects. Character and Block special devices are treated as private Helios objects, and cannot be used from Helios. Symbolic links, sockets, and fifos are not supported in the current release.

## 2.3 The Error Logger

The I/O Server contains a device, */logger*, which may be used by Helios programs for error output. By default, all data that is sent to the logger device is diverted to the Server's own window, but an alternative destination may be specified by the user. When running under SunView this is done by clicking a mouse button on the Logger cycle; on a dumb terminal it is done by using the key sequence, <*hot key*>l. A Server status request will display the current logging destination.

If the logging destination is a file, or both file and window, any data sent to */logger* is appended to the end of a logfile. This data may be read from Helios using standard commands; for example,

```
cat /logger
```

If you wish to empty the logger file, you can do this from Helios using the command-line:

```
rm /logger
```

Alternatively, when the Server exits, any data written to the logger will be preserved in the logfile, whereupon you may examine it at your leisure. The file will be cleared when the Server is run again.

There are two entries in the *host.con* file that control the behaviour of the error logger. The first entry, *logfile* = *<filename>*, specifies the file that is to be used to store logging output; the default is *logfile* in the current directory. The second entry is *logging_destination*, which can be set to *screen*, *file*, or both, and controls the initial logging destination. The following example entries would cause any data written to the logger to go to a file called logbook.

```
logfile = logbook
logging_destination = file
```

## 2.4 Multiple Links

On many Unix-hosted transputer systems the host has multiple link adapters into a transputer network or into different transputer networks. If each link adapter is connected to a different root transputer, then it is possible to have multiple users running Helios. This combination of link adapter, root transputer, and possibly some additional transputer network is known as a **site**. For example, some transputer boards have four sites, allowing four users to run Helios at the same time. Any site not currently used for Helios can be used for other software, such as TDS.

The I/O Server must be able to interact with the link adapter, directly or indirectly. If the I/O Server runs on the host with the link interface, it can access the link directly. If a user wishes to access transputers in a remote machine over the Ethernet (or other local network), and still have the

benefits of real windows etc., then the I/O Server must run on his own machine and interact with the link via a link daemon known as **Hydra**. Communication between the I/O Server and Hydra takes place using TCP/IP sockets over the Ethernet.

As each transputer site may be different, there must be some way for the user to specify which site they will use. This can be done by adding a suitable entry into *host.con*, in the form *site = 0*, *site = 1*, etc. Note that the sites are simple integers which are mapped onto the actual hardware by the Server. On a Sun with a Transtech board, site 0 corresponds to the link device */dev/nap0*, site 1 corresponds to */dev/nap1*, and so on.

The Telmat workstation can incorporate up to eight different CPU modules, each accepting up to eight transputer boards. As a result, 8x8 ITFTP32 link interfaces can run at the same time. The device link name and the site number are declared by the following formula:

the device link name = */dev/link<x>_mt<y>*

where

<x> = the interface board number
(*x* can take the values: 0, 8, 16, 24, 32, 40, 48 or 56)

and

<y> = the CPU module number
(*y* can take the values: 0, 1, 2, 3, 4, 5, 6 or 7)

the site number = (<x> / 8) + (8 * <y>)

For example, the board plugged at address 24 of CPU module number 2 has the name */dev/link24_mt2*, and the site number to declare in the *host.con* file is 19.

If no site is specified in the *host.con* file, the Server will choose any available site. At some future stage it is intended to provide a dialogue between the Server and the user to allow the latter to choose a site interactively.

The configuration file controls whether the Server communicates directly with a link device (in which case it has to run on the same host), or that it interacts with the link daemon, Hydra. The former is significantly more efficient as it avoids the communication overhead within the Unix world, but it does not allow remote access. The relevant entry can be

*box = <hardware name>*

or

*box = remote*

If the entry specifies a particular piece of hardware, NTP1000 for the Transtech Sun board, ITFTP32 for the Telmat board, then the Server will interact directly with the link; if the entry specifies *remote*, then the Server will interact with the link daemon, Hydra.

If the box is specified as remote, the Server will interact with Hydra over a TCP/IP socket. This socket can be within the Unix family or within the internet family, and is controlled by the *family_name* entry in the configuration file; a setting of *AF_UNIX* or *AF_INET* should be used as appropriate. With the Unix family it is necessary to specify a name within the Unix filing system for the socket. For example,

```
family_name = AF_UNIX
socket_name = my_socket
```

uses a Unix socket called my_socket in the current directory. The default family is AF_UNIX, and the default socket name is *hydra.skt*.

If the socket family is AF_INET, the Server will use the normal networking routines to connect to the Hydra daemon. It needs to know the network name of the machine on which Hydra is running, and this is specified by using the *hydra_host* entry in the configuration file.

```
hydra_host = sp2sun1
```

This host name must correspond to an entry in the */etc/hosts* file. Given the host address, the Server uses the network routines **gethostbyname()** and **getservbyname()** to obtain a socket identifier. This requires the installation administrator to enter *hydra* in the file, */etc/services*, when Helios is installed. The file should be modified on each of the machines which is likely to be used for running Hydra or the I/O Server. The entry should like this,

```
hydra 1234/tcp
```

where 1234 is any socket number not used by other services.

It is possible that Hydra may be unable to accept a new connection immediately. This is particularly true if the system is heavily loaded by one or more users who are booting at the same time. Under these circumstances the Server will display the message

```
Hydra is busy...
```

and retry after a short delay. The number of retries is specified in the *host.con* file by the entry, *connection_retries*. The default value is 5.

The protocol used between Hydra and the Server is independent of the hardware, and hence the two programs may run on completely different machines. For example, it is possible to run Hydra on a Sun-4 and the I/O Server on a Sun-3, or vice versa.

## 2.5 Hydra

The link daemon, Hydra, is a separate program which is normally under the control of the installation administrator, and is run automatically when the host boots up. A separate monitoring program, **hydramon**, can be used to interrogate Hydra to determine which sites are currently in use, and by whom. In addition, it can be used to disconnect a particular Server and release the site; this disconnection occurs immediately, and may result in the loss of data, so access to hydramon is normally restricted. Hydramon also allows sites to be released, which means that Hydra will no longer allow access to those sites, and allows sites to be used again.

Both Hydra and the hydramon program read the configuration file ***hydra.con***. A typical *hydra.con* file might look something like this:

```
host            = SUN
box             = B011
hydra_host      = sp2sun1
#family_name    = AF_UNIX
#socket_name    = my_socket
family_name     = AF_INET
connection_delay = 25
#all_sites
nap0
#nap1
nap2
#nap3
```

The *hydra.con* file closely resembles the *host.con* file, although there are fewer options. It specifies the type of host and transputer network, and the network address of the host. This network address should correspond to an entry in */etc/hosts*. Like the Server, Hydra can use either Unix or internet sockets, and for internet sockets the system administrator must add Hydra to the list of available services in */etc/services*, using any free socket number. *Connection_delay* specifies a delay in seconds between accepting new connections from Servers; the bootstrap process requires a considerable amount of input/output, so having multiple users booting up within a short time of each other may overload Hydra.

The final entries in the file specify which sites are to be used by Hydra. It is possible for Hydra to use all available sites, or only a selected number of sites. The latter option is useful to allow users to access particular sites without going via Hydra, or even to run software other than Helios on these sites. Hydra only locks sites that are currently running Helios, so it is possible to access a site directly even if it is one of the sites accessible via Hydra.

For reliable operation, Hydra must put the link devices into non-blocking mode. Given the lack of memory protection in the transputer hardware, it is perfectly feasible for the transputer to crash in the middle of link traffic, in which case Hydra should be able to recover rather than hang on a read or write. If the device does not support non-blocking mode it is possible for every Helios session to hang because of a single transputer crash, and Hydra will display a warning to that effect.

## 2.6 Other 'host.con' Configuration Entries

There are a number of other entries in the *host.con* configuration file which can be changed by the user; examples of these are listed below.

`message_limit = 30000`

This specifies the maximum size of the data vector that is used to transfer messages between Helios and the I/O Server. On a Unix system it is very expensive to have small message limits, because transferring data to and from the link involves switching between the I/O Server and the operating system. Having smaller limits can avoid problems on some other machines, but is unlikely to do so under Unix. The maximum message limit is 64000.

`root_processor = /00`

Under Helios, every processor has a name. The name of the root processor is controlled by the I/O Server, and cannot change while Helios is running. The default name is /00, but it can be changed via the configuration file.

`io_processor = /sun`

The I/O processor behaves just like a transputer within the Helios network, and hence it too must have a network name. The default name is /IO, but this can be controlled by the user.

`transputer_memory = 0x200000`

On a normal system, Helios itself determines the amount of memory attached to the transputer. Problems may be experienced if the main transputer memory is followed immediately by video memory or other hardware, as Helios will attempt to use this memory as well. By specifying the amount of memory using this entry in the configuration file, Helios will not attempt to use the special areas. You should ensure that this option never specifies a larger amount than that which is actually available; if you ignore this warning, Helios will attempt to use non-existent memory. The example entry shown above specifies that the system has two megabytes of memory (using hexadecimal).

`bootlink = 1`

On most transputer hardware the I/O processor is connected to link 0 of the root transputer, so Helios assumes that this will always be the case. If your hardware is different you should specify an alternative by applying this entry.

Here is an example host.con file:

```
host            = SUN
box             = B014
#site           = 0

#box = remote
#family_name = AF_UNIX
#socket_name = silly
family_name = AF_INET
hydra_host = sp2sun1
connection_retries = 10

message_limit = 60000
helios_directory    = /home/sp2sun1/user1/helios
system_image  = ~/lib/nucleus
bootfile = ~/lib/nboot.i

logfile = logbook
logging_destination = screen

#transputer_memory = 0x100000
bootlink = 1
#root_processor = /00
#io_processor = /sun

Server_windows
#server_windows_nopop
escape_sequence =ku
switch_forwards_key = kl
switch_backwards_key = kr
#status_key = kd
#debugger_key = kl
exit_key = kd
#reboot_key = kr
#refresh_key = kd
```

With this configuration file the Server would interact directly with a link adapter rather than going via Hydra, and it would choose any free site. On a dumb terminal the cursor keys would be used for escape sequences. The other options are all standard.

# Chapter 3

# Helios Version 1.1

This chapter summarises some of the major additions and changes in Helios version 1.1.

## 3.1 Job Control in the Shell

When a command is executed in the background using the shell metacharacter &, it is referred to as a job. Whenever the shell creates a job it assigns it a job number and a process identification number (process id). It then displays these identifiers to the user in a line of the form:

*[<job_number>] <process_id>*

A complete list of all the current jobs and their associated numbers are maintained by the shell, and can be viewed with the **jobs** command.

When a job has been created there are two operations which the user can perform on it: it can be brought to the foreground or it can be terminated. For either operation you will need to know the job's name.

| Job Name | Description |
|---|---|
| *%<job_number>* | The job with the specified job number. |
| *%%* | The current job. |
| *%+* | The current job. |
| *%-* | The previous job. |

Each of the character sequences shown above can be supplied as an argument to **kill** or **fg** to terminate the specified job or bring it to the foreground. The character sequences can also be entered directly into the

shell, and will have the same effect as if they were supplied as arguments to **fg**. Full details of **kill** and **fg** are given in Section 3.7, "New Commands in Helios 1.1".

Whenever a job terminates, the shell removes the job from its internal list and displays a message of the form:

*[<job_number>] <exit_status> <command>*

The *exit_status* in this message takes one of three forms. If the command terminated successfully then the message is "Done". If the command exited with a non-zero exit code, the message takes the form "Exit *n*", where *n* is the the argument which was passed to the exit function. If the command was terminated by a signal, then the message describes that signal. For example, if job number 1 terminates because of stack overflow, the shell displays:

```
[1]  Stack Overflow  myprog
```

## 3.2 Alias Server

There is a new server, *lib/alias*, which allows the user to assign an alias to a directory name. The syntax for this server is as follows:

*/helios/lib/alias <name> <directory>*

This command should always be run in the background, either by appending the metacharacter & when entering the command into the shell, or by using the **run** command in the startup file, *initrc*. A typical application of this command is

```
/helios/lib/alias etc /helios/etc &
```

which causes all future references to */etc* to be interpreted as */helios/etc*. This feature is particularly useful for mapping the Helios directory structure onto that of another operating system; makefiles and shell scripts can then be ported with very little modification.

## 3.3 Batch Server

The Batch Server provides a program scheduling service. It uses a job description language to define the program which is to be executed remotely. It allows the user to specify the time at which the program will be executed, its environment, and the interval at which it should be rescheduled (if required).

The syntax of the job description language is given here in BNF:

*<batch_description> ::= <job_description> <batch_description>*

*<job_description> ::= TFname [arguments] '{' <parameters> '}'*

```
<parameters>  ::=  'START'    <start_time>   <parameters>
                |  'REPEAT'   <repeat_delay> <parameters>
                |  'PARENT'   <name>         <parameters>
                |  'STATUS'   <status>       <parameters>
                |  'PRIORITY' <priority>     <parameters>
                |  'SYSTEM'                  <parameters>
                |  'ENVIRON'  <envv>;        <parameters>
                |  'OBJECTS'  <objv>;        <parameters>
                |  'STREAMS'  <strv>;        <parameters>
```

The parameters used in the job description are described below.

*START*

The START parameter gives the time at which the job (task force) should be started (*start_time*). If no start time is given, or it is specified as 0, then the task force is started immediately. The format is

*<start_time> ::= [day]':'[month]':'[year]':'[hour]':'[minutes]':'[seconds]*

or

dd:mm:yyyy:hh:mm:ss

where *dd* is the day of the month, *mm* is the month of the year and *yyyy* is the year. The time is given in 24-hour format by *hh:mm:ss*. For example,

```
26:11:1989:10:00:00
```

specifies a time of 10am on the 26th November 1989.

If any of the time fields are replaced by a tilde ('~') then the start time given is added to the current time. Thus ~:~:~:2:20:00 will start the job 2 hours 20 minutes from the job submit time. If any date field is substituted by '~' then the current value is assumed. So if the current date is 26:11:1989:10:00:00, then a start time of 28:~:~:~:~:~ will run the job on the 28th November at 10:00.

*REPEAT*

The REPEAT parameter specifies the delay between consecutive instances of the job (*repeat_delay*).

*<repeat_delay> ::= [hour]':'[minutes]':'[seconds]*

Again, the the same format is used for the time (*hh:mm:ss*). If no repeat delay is given then the task force is only executed once.

*STATUS*

The STATUS field gives the job status.

*<status>*

The default value of this field is 0, which means that the job is mortal and will be deleted on error. If the value is 1 it means that the job is immortal and will be automatically rescheduled on error. A value of 2 means the job will be deleted immediately.

*PRIORITY*

The PRIORITY field is defined to allow priorities at some future date. It is not currently used.

*<priority> ::= number*

*SYSTEM*

The SYSTEM parameter defines that the job is a system service and no environment will be sent to the task force when it starts executing. Unless this keyword is given, the job is a user job and is always sent an environment.

*ENVIRON*

The ENVIRON keyword is used to specify an environment string to the task force.

*<envv> ::= name <envv>*

*OBJECTS*

The OBJECTS keyword is used to specify objects, which must exist, to the task force to be run. These objects are also passed in the environment.

*<objv> ::= object_name <objv>*

*STREAMS*

The STREAMS keyword is used to specify open streams which are to be passed as the environment for a task force. Note that most C programs require *stdin*, *stdout* and *stderr* to be defined within their environment.

*<strv> ::= stream_name <strv>*

*TFname*

TFname is the name of any executable task force (either an executable object or a CDL object); however, the full context of the task force object must be given. Here is an example script to illustrate the job description language:

```
/helios/lib/fastfiler -b -k/cache/bin {
                start  10:9:1990:14:20:0
                status 1
                streams /null /null /helios/error/filelog;
}
```

```
/helios/bin/garbage_collect {
                repeat 1:0:0
                status  0
                environ  NOCACHE DEADBLOCK;
                system
}
```

Having created the job description file, you can submit it with the **runb** system utility. For example,

```
runb testscript.jb
```

creates the job and then passes it to the Batch Server for the component programs to be executed as required. If the job description is syntactically incorrect, **runb** will return with an error.

## 3.4 Fault Library

The Fault library is used to search a fault database for matching fault and error codes. There are two ways of using it: via **Fault()**, which searches the standard fault database in */helios/etc/faults*; or via the routines **fdbopen()**, **fdbrewind()**, **fdbfind()** and **fdbclose()**. Templates for all these functions are to be found in *fault.h*.

### 3.4.1 Fault

The first argument to **Fault** is an error code. If the value is less than zero, it is interpreted as a Helios error code. If the value is greater than zero, but less than or equal to the highest POSIX error code, it is treated as a POSIX error code; otherwise, it is treated as a Helios function code. The second and third arguments of this function are a message buffer and its size. The message is added into this buffer as a null terminated string, and will be truncated if it does not fit.

### 3.4.2 Low Level Routines

The procedures described here allow a private fault database to be used to generate messages. The function **fdbopen** attempts to open the named file as a fault database and will return NULL on error; **fdbclose** closes a fault database. As the fault database is only searched forwards, **fdbrewind** repositions the search point at the start of the file.

The function **fdbfind** searches the fault database for an entry. The second parameter is the name of the fault class to be searched. The third parameter contains the code to be searched for; only the bits of this value described in the class entry's mask field will be compared. The remaining two parameters describe a buffer; the message corresponding to the code will be concatenated onto the end of the existing buffer contents, but only if the remaining space in the buffer is large enough.

### 3.4.3 Fault Database Format

A fault database is an ASCII file organised in lines. Any line beginning with "#" is ignored. Numerical values may be given in decimal or in hexadecimal (hex). Hex values must be preceded by "0x". A line beginning with a '!' is a class description: the first field is the class name, the second field is a mask which indicates the bits that class occupies, and the optional third field gives the C header prefix. A class ends with a line containing just "!!".

Within a class each line consists of a code name, a code value, and an optional message string. If the message string is not present, then the code name is used.

## 3.5 Technical Changes

The Helios V1.1 kernel is faster than previous versions, particularly when it is passing messages through a processor from one link to another. This kernel also contains support for attaching routines to the Event line. Other kernel features include a processor performance monitor and a port table garbage collector.

The format of function codes has changed slightly to include a Retry field, which is used by the Processor Manager to maintain a confidence level associated with each name in the name table. When this level drops below a certain threshold the name is removed from the name table, forcing a new distributed search for it. This provides an extra level of recovery in the face of processor and link crashes.

Module init routines are now called twice, once with a second argument of 0 and once with a second argument of 1 (this argument used to be undefined).

The arbitrary limit of 20 open files has been lifted in the POSIX library; any number are now allowed. However, the C library limit remains, largely as a result of compatibility issues involved in the implementation.

The new Pipe Server provides bi-directional communication between processes. A new server, */pipe*, is used to support this and should be used in preference to */fifo* for inter-task communication. The shell and TFM now use */pipe* by default. Pipes, unlike fifos, have no internal buffering: communicating processes which are connected via the Pipe Server interact directly; the only buffering available is that supplied by the runtime system.

When booted, the kernel now ensures that it synchronises internally, and with its parent, before continuing. For this reason **BootLink()** has been moved to the system library, so any programs that call it should at least be re-linked.

The C compiler has been further optimised since the last release; in particular, branch chains are now eliminated along with unreachable code, and some peep-hole optimisation is performed at code generation. The head label of all loops is now aligned to a word boundary to make maximum use of the instruction fetch. All instances of **mul** have been converted to **prod**, since Helios makes no use of the error flag.

The previous version of the C compiler used the wrong rounding modes for the floating point conversion routines; this has now been fixed. The T800 floating point libraries are now correctly re-entrant, and the full set of rounding modes are supported.

Open streams passed to a program in its environment (stdin, stdout, etc.) are now not actually opened until first used. In doing this we have found that program startup is now marginally faster than it was before.

## 3.6 Component Distribution Language

This section describes the enhancements that have been made to the Component Distribution Language (CDL), for Helios 1.1. These changes are designed to simplify the description of complex, multi-component task forces. Compatibility with Helios 1.0 has been maintained as much as possible, the only change which may cause problems is the introduction of a precedence for parallel constructors.

### 3.6.1 Component Attribute Code

An additional attribute, 'code', is supported inside component declarations. Code is used to introduce a filename which specifies the actual piece of code to which the component refers. The effect of this is to remove the direct association between the name of a component and the code that it executes; for example, several components can now be defined, each having different resource requirements and the same code. For compatibility with Helios 1.0, if no 'code' attribute is specified, the name of the code defaults to the name of the component. For example, the following CDL script:

```
component musthaveT414
{
  code myprog;
  processor T414;
  memory 100000;
}

component musthaveT800
{
  code myprog;
  processor T800;
}
```

```
musthaveT414 ^^ musthaveT800 ^^ myprog
```

will execute three copies of 'myprog' in parallel. The first copy must execute on a T414 with at least 100000 bytes of memory, the second on a T800, and the third has no preference. Each component refers to the same code, which the CDL compiler locates by using the PATH environment variable. Note that memory requirements must be specified in decimal.

### 3.6.2 Subscripted Component Declarations

Component declaration names can now be followed by one or more subscript names. Each subscript name is separated from the next by a comma, and the entire list is enclosed in square brackets. Within the body of the declaration, any stream name can include a list of subscript expressions which reference these subscript names. For example, the following component declaration:

```
component filter[i]
{
  code cat;
  streams <| pipe[i], >| pipe[i+1];
}
```

uses the subscript *i*. This name is used subsequently in two subscript expressions within the body of the declaration; each expression is evaluated when the component is referenced. Whenever a component declaration is referenced within a task force definition, subscript values can be provided which are bound to the subscript names within the declaration. The aim of this is to be able to declare a component which, depending on subscript values it is referenced with, communicates on different streams. Many structures, such as arrays, can be defined by declaring a single component which has subscripted stream names. The structure is then defined by referencing this component with different subscript values.

Command names used with task force definitions may now be followed by a list of subscript values that are enclosed in braces and separated by commas. These subscripts may also be expressions, but for the purposes of this explanation they are assumed to be values. For example, the task force definition:

```
filter{0} ^^ filter{1} ^^ filter{2}
```

defines a task force which comprises three components. Each component refers to the component declaration for *filter* (given above), but in each case different subscript values are passed into the declaration. The important point is that stream names with different subscript values refer to different streams. Thus *filter{0}* reads from the stream *pipe{0}* and writes to the stream *pipe{1}*, *filter{1}* reads from stream *pipe{1}* and writes to stream *pipe{2}*, and *filter{2}* reads from stream *pipe{2}* and writes to stream *pipe{3}'*. As it stands the task force definition is incomplete because the stream *pipe{0}* has no component writing to it and the stream *pipe{3}* has no component reading from it. For a task force to be complete, each stream that it uses must have one component reading from it and one writing to it. A complete task force definition might be as follows:

```
cat >|pipe{0} ^^ cat <| pipe{2} ^^
filter{0} ^^ filter{1} ^^ filter{2}
```

which is a pipeline made up of five **cat** commands.

### 3.6.3 Replicators

In the previous version of CDL, replicators preceded constructors. For example,

```
ls [3]| cat
```

is equivalent to

```
ls | cat | cat | cat
```

thus replicating the constructor and its preceding command three times. This is still supported, but Helios 1.1 also supports a new variation in which the replicator directly follows the constructor. This new form of replication does not define the constructor used to communicate with the preceding construction. For example,

```
|[3] cat
```

is equivalent to

```
cat | cat | cat
```

The first example in this section can therefore be written as:

```
ls | (|[2] cat)
```

This also affects the use of the interleave constructor (previously called the farm constructor). The usual method of defining a farm construct,

```
control [3]||| worker
```

is still valid and is equivalent to:

```
control <> lb 3 (<> worker, <> worker, <> worker)
```

The load balancer command, **lb**, is inserted automatically by the CDL compiler and is given the job of interleaving the input and output from replications of *worker*. The particular implementation of **lb** supplied in the standard Helios *bin* directory attempts to balance the workload on each of the workers by way of a packet protocol; this can be overloaded as described in *The Helios Operating System*, Section 7.5. We are not concerned here with the implementation details of **lb**, in fact we can ignore them altogether and just think of the input and output of the workers as being interleaved.

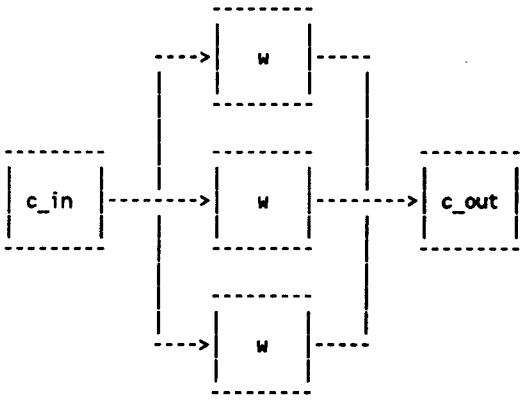
The interleave constructor can also be used with the new syntax for replicators. Using the new syntax, you would write the previous farm construct as:

```
control <> (|||[3] worker)
```

This is more flexible because you can now make use of an interleave construct as part of a pipeline. For instance,

```
control_in | (|||[3] worker) | control_out
```

defines a structure which looks like:




**Note:** Currently, the interleave constructor cannot be used as a binary operator but must always be used in conjunction with a replicator.

### 3.6.4 Multi-Dimensioned Replication

Whereas Helios 1.0 only allowed a single count within replicators, Helios 1.1 allows multi-dimensioned replication by accepting and interpreting a list of counts, separated by commas. For example,

```
^^[2,3] node
```

is equivalent to

```
node ^^ node ^^ node ^^ node ^^ node ^^ node
```

The use of multi-dimensioned replicators will become apparent in the next section.

**Note:** Multi-dimensioned replication of pipes and subordinates does not produce a multi-dimensioned structure, but rather a one-dimensional pipeline or bi-directional pipeline just as if a single-dimensioned replicator was used. This may change in a future version and for this reason we recommend that multi-dimensioned replicators are only used with the simple parallel and interleave constructors. This should not present a problem as all structures should be achievable with a full component declaration.

### 3.6.5 Iteration Names

An iteration name may be associated with each dimension of a replicator. The replicated construction may contain command name subscript expressions and stream name subscript expressions involving these iteration names. When the replication is expanded, each successive replication is passed values for the iteration name from 0 to one less than the number of replications. The syntax for introducing iteration names is to precede the replication limit by the name followed by a '<' symbol. For example,

```
|[i<3] filter{i}
```

is equivalent to the task force

```
filter{0} | filter{1} | filter{2}
```

Thus the earlier example can now be written:

```
cat >| pipe{0} ^^ (|[i<3] filter{i}) ^^ cat <| pipe{2}
```

The scope of each iteration name is the entire construction being replicated. Since these constructions may contain further replication this scope may contain 'holes' where the same iteration name is redefined. A reference to an iteration name obtains the value of the most closely nested one of that name. The scope rules are similar to the scope rules of variables in a block-structured language. For example,

```
^^[i<2,j<2] (a(i,j) ^^ |[i<3] b(i,j))
```

is equivalent to

```
(a(0,0) ^^ b(0,0) | b(1,0) | b(2,0)) ^^
(a(0,1) ^^ b(0,1) | b(1,1) | b(2,1)) ^^
(a(1,0) ^^ b(0,0) | b(1,0) | b(2,0)) ^^
(a(1,1) ^^ b(0,1) | b(1,1) | b(2,1))
```

The 'i' and 'j' used with the 'a' component refer to the iteration names defined in the first replicator, as does the 'j' used with the 'b' component. The 'i' used with the 'b' refers to the 'i' defined in the second replicator.

The value of an expression involving iteration names can be passed as an argument to a component. The expression is placed in the list of arguments to the component in the task force definition and is distinguished from a normal argument by preceding it by a % character. For example,

```
^^[i<3] node(i) %i+1
```

is equivalent to

```
node(0) 1 ^^ node(1) 2 ^^ node(2) 3
```

**Note:** Whereas stream name subscript expressions are used to form complete stream names, command name subscript expressions, once used to expand the referenced component declaration, are of no further significance.

### 3.6.6 Precedence of Constructors

In Helios 1.1 each constructor has a unique precedence, as opposed to the common precedence of all constructors in Helios 1.0. The constructors have the following order of precedence, in ascending order from left to right:

```
^^  |||  |  <>
```

For example,

```
a <> b | c
```

is equivalent to

```
(a <> b) | c
```

rather than

```
a <> (b | c)
```

(note the use of parentheses to override the precedence) and

```
a <> b | c ^^ d | e
```

is equivalent to

```
((a <> b) | c) ^^ (d | e)
```

This affects the way in which streams are allocated and the configuration of the resulting task force.

### 3.6.7 Automatic Allocation of Streams

The rules covering the automatic allocation of streams to task forces are given below.

A simple parallel constructor, ^^, defines no communication between its operands.

A pipe constructor, |, defines a single communication between its operands. In general, for the task force

```
A | B
```

file descriptor 1 of A is connected to the file descriptor 0 of B. Note that in this and in subsequent examples, A and B may themselves be task forces.

A subordinate constructor, <>, defines a pair of communications between its operands. For example, in

```
A <> B
```

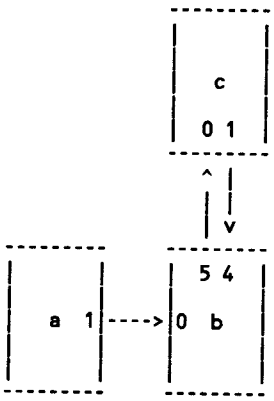
file descriptor 1 of B is connected to the first auxiliary input of A (this is defined to correspond to file descriptor 4 at the POSIX level), and the first auxiliary output of A (POSIX file descriptor 5) is connected to the file descriptor 0 of B.

The order of allocation of streams for a task force is defined by the precedence of the constructors. Thus for

```
a | b <> c
```

first the streams for the subordinate constructor are allocated and then the streams for the pipe constructor. Streams for constructors of the same precedence are allocated from left to right.

Once a file descriptor of a component has been overloaded it becomes a hidden internal stream and cannot subsequently be overloaded. So, for the previous example, file descriptor 0 of component 'c' is initially overloaded by the allocation of streams for the subordinate constructor and consequently is not further overloaded by allocation of streams for the pipe constructor. The example produces a structure that looks like:




where each box represents a component; a line connecting two boxes represents a stream, with the arrow giving its direction and the number representing the file descriptor.

Stream allocation within an auxiliary list is a special case, because each of the constructors within the list has the same left hand operand, as in the following example.

```
master (<> slave1, | slave2, <> slave3)
```

In this case, both subordinate constructors and the pipe constructor have the same left operand, 'master'. Each such constructor in an auxiliary list uses successive auxiliary streams of its common component, with the proviso that they are allocated in pairs. Remember that the first auxiliary stream corresponds to POSIX file descriptor 4. An input is allocated on an even-numbered file descriptor and an output on an odd-numbered file descriptor. So, in this example, 'master' communicates with 'slave1' on file descriptors 4 and 5, 'slave2' on file descriptor 7, and 'slave3' on 8 and 9.

### 3.6.8 Subscript Expressions

Subscript expressions are written in standard arithmetic format and may contain integers, subscript names, binary operands, unary operands, and parentheses. The binary operands that are supported here are as follows:

+ addition
- subtraction
* multiplication
% remainder

+ and - are also supported as unary operands.

### 3.6.9 CDL Compiler

In addition to supporting the enhancements described in this document, the latest version of the CDL compiler has a few new features, which are outlined here.

As mentioned earlier, the latest compiler validates the use of streams. Every stream used by a task force must have one reader and one writer and any other combination will result in a compilation error.

### 3.6.10 CDL Scripts

This section contains some example CDL scripts. In each we are not interested in the details of the application code but merely in how to generate the correct structure in terms of components and connecting streams.

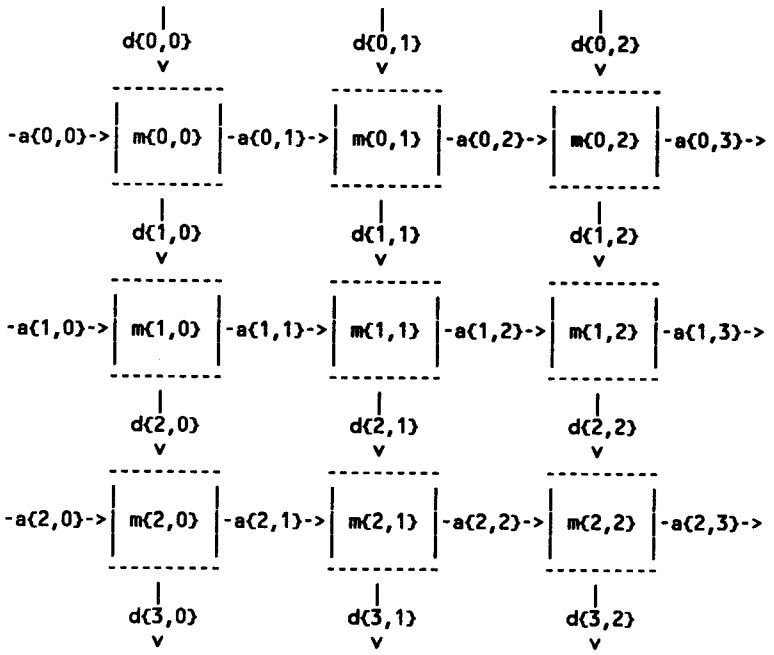
In this first example, we wish to define a task force consisting of 10 components, where each component is a filter, reading on file descriptor 0 and writing on file descriptor 1. We require these streams to be connected to form a 'ring' of components. To achieve this we first declare the following component:

```
component node[i]
{
  streams <| pipe[i], >| pipe[(i+1)%10];
}
```

The actual task force definition simply consists of a replicated structure of 10 invocations of 'node' running in parallel.

```
^^[i<10] node{i}
```

For our second example, we wish to generate a two-dimensional matrix of components, where each component has two inputs and two outputs. A diagram is the easiest way to show what we require.




Sensible subscript values for components and stream names is vital in defining a complex task force; it enables the stream name subscripts of an expression to be expressed in terms of the subscript values of the component. This is exactly what we have done in the following component declaration:

```
component mult[i,j]
{
  streams <| across{i,j}, >| across{i,j+1},
          <| down{i,j},   >| down{i+1,j};
}
```

Here 'mult' refers to the 'm' in our diagram, 'across' to 'a', and 'down' to 'd'. The task force definition replicates this component, passing the required subscript values, and looks like:

```
^^[i<3,j<3] mult(i,j)
```

**Note:** This example does not define a complete task force.

### 3.6.11 New CDL Syntax

This section gives an updated formal definition of the CDL syntax.

```
<script>       ::= { <declaration> } <taskforce>

<declaration>  ::= 'component' <name> <sub-names>
                   '{' { <attribute> [';'] } '}'

<attribute>    ::= 'code' <name>
                 | 'processor' <ptype>
                 | 'puid' <name>
                 | 'attrib' <attriblist>
                 | 'memory' <size>
                 | 'streams' <streamlist>

<ptype>        ::= 'T414' | 'T800' | 'ANY'

<attriblist>   ::= <attrib> { [','] <attrib> }

<attrib>       ::= <name> [ '[' <number> ']' ]

<streamlist>   ::= <stream> { [','] <stream> }

<stream>       ::= <mode> <name> [ <sub-exprs> ]

<mode>         ::= '<' | '>' | '>>' | '>|' | '<|'

<taskforce>    ::= <interleave> { '^^' <interleave> }
                 | '^^' <replicator> <interleave>
                 | <replicator> <construction>

<interleave>   ::= <pipeline>
                 | '|||' <replicator> <pipeline>
                 | <replicator> <construction>

<pipeline>     ::= <subordinate> { '|' <subordinate> }
                 | '|' <replicator> <subordinate>
                 | <replicator> <construction>

<subordinate>  ::= <command> { '<>' <command> }
                 | '<>' <replicator> <command>
                 | <replicator> <construction>

<command>      ::= <simple cmd> [ <auxlist> ]
                 | '(' <taskforce> ')'  [ <auxlist> ]
```

```
<construction> ::= '^^' <interleave>
                 | '|||' <pipeline>
                 | '||' <subordinate>
                 | '<>' <command>

<auxlist>      ::= '(' <aux> { ',' <aux> } ')'

<aux>          ::= '||'  <taskforce>
                 | '||<' <taskforce>
                 | '<>' <taskforce>

<simple cmd>   ::= <name> [ <sub-exprs> ]
                   { ( <arg> | <stream> ) }

<mode>         ::= '<' | '>' | '>>' | '<|' | '>|'

<sub-names>    ::= [ <name> , { <name> } ]

<sub-exprs>    ::= '{' <sub-expr> { <sub-expr> } '}'

<sub-expr>     ::= <unary-expr> { <binary-op> <sub-expr> }

<unary-expr>   ::= <number>
                 | <name>
                 | <unary-op> <sub-expr>

<binary-op>    ::= '*' | '%' | '+' | '-'

<unary-op>     ::= '+' | '-'

<replicator>   ::= '[' <dimension> { , <dimension> } ']'

<dimension>    ::= [ <name> '<' ] <number>

<name>         ::= sequence of printable characters

<number>       ::= sequence of decimal digits
```

## 3.7 New Commands in Helios 1.1

This section provides a detailed description of the commands: **boot, c, dlink, elink, fg, kill, lbpcat, lcontrol, map, netversn, reset, run, runb, startns,** and **tcp.**

# boot

**Purpose:** To boot the given subnetwork.

**Format:** *boot <subnet_name>*

## Description:

**boot** first resets the subnetwork *subnet_name* (this may just be a single processor), enabling the Network Server to boot it. This command will not return until the whole subnetwork is booted. For example,

```
boot /net/clustera
```

boots the subnetwork /net/clustera.

**Note:** There is a 60 second timeout on this command, so **boot** may return with a timeout error when booting very large subnetworks, although the subnetwork will still be booted.

# c

**Purpose:** To compile and link a program.

**Format:** *c [opts] <filename> [<filename> ...]*

## Description:

The compiler driver **c** is used to compile and link a program. It takes a list of files and decides what to do with them according to their suffix. The filename suffixes that are supported by the compiler driver are as follows:

| Suffix | Meaning |
|---|---|
| .a | Macro assembly language (AMPP source) |
| .c | C language |
| .f | FORTRAN language |
| .o | Object file format |
| .s | Assembly language |

If no other arguments are given, the program compiles the programs for the languages specified as .c or .f, assembles any .s files and then links all the resulting binaries, along with any supplied .o files, into an executable program called a.out. A large number of options can be used to alter the behaviour of the program, as follows:

| Option | Action |
|---|---|
| -b | Don't link with standard libraries (*fplib* and *fpclib*). |
| -c | Compile/Assemble only, don't link. |
| -d<*name*> | Specify output file name for library *.def* compilations. |
| -e[6\|7] | Enforce FORTRAN standard. |
| -h<*val*> | Specify heap size of program. |
| -j | Join objects but don't link. |
| -l<*name*> | Link with standard library <*name*> (*/helios/lib/<name>lib.def*). |
| -m | Compile code for libraries. |

| Option | Action |
|---|---|
| -n | Don't actually execute commands (implies -v). |
| -n<*string*> | Specify object name of program. |
| -o <*name*> | Specify output name (default *.o or "a.out") |
| -p | Compile code for profiling. |
| -q<*string*> | Enable compiler debugging features. |
| -s<*val*> | Specify stack size of program. |
| -t | Compile code for tracing. |
| -v | Verify command being executed. |
| -w[acdfpsvz] | Suppress warnings. |
| -A<*flag*> | Pass <*flag*> direct to linker. |
| -B | Do not link with any libraries. Do not perform **objed**. |
| -C | Perform array bound checking (F77). |
| -D<*name*> | #define <*name*> (C) |
| -D<*name*>=<*val*> | #define <*name*> to be <*val*> (default <*val*> is 1) (C). |
| -F[fghmsv] | Enable compiler features ('s' turns off stack checking and 'g' suppresses insertion of function names in code) (C). |
| -I<*dir*> | Specify a directory to be searched for #include files. |
| -L<*name*> | Link with standard library <*name*> (*/helios/lib/<name>.def*). |
| -M<*name*> | Produce map file <*name*> (F77). |
| -O | Optimise code; perform full link. |
| -S | Produce textual assembler output in *.s; don't link. |
| -V | Pass on verbose flag to executed commands. |
| -T[4\|8] | Specify Transputer type. |
| -W<*val*> | Specify warning level (F77). |
| -X<*val*> | Specify cross reference width (F77). |
| -help | List this message. |

# dlink

**Purpose:** To disable the given link.

**Format:** *dlink <subnet_name> <link_number>*

## Description:

**dlink** disables the given link and sets it into dumb mode; for example:

```
dlink /net/clustera/04 1
```

# elink

**Purpose:** To enable the given link.

**Format:** *elink <subnet_name> <link_number>*

## Description:

**elink** is the opposite to **dlink**: it enables the given link. This will set the link into intelligent mode and attempt to enable it:

```
elink /net/clustera/04 1
```

# fg

**Purpose:** To bring a job to the foreground.

**Format:** *fg [<job_name>]*

## Description:

This shell command brings the specified job, *job_name*, to the foreground. If no argument is supplied, **fg** uses the current job. The following character sequences can be supplied as arguments, and have the meanings shown:

| Job Name | Description |
|---|---|
| %<*job_number*> | The job with the specified job number. |
| %% | The current job. |
| %+ | The current job. |
| %- | The previous job. |

# kill

**Purpose:** To terminate the specified job.

**Format:** *kill <job_name> | <processid>*

## Description:

This shell command is used to terminate the specified job; the job can be identified by either its job name or its process identification number. The following character sequences can be supplied as arguments, and have the meanings shown:

| Job Name | Description |
|---|---|
| *%<job_number>* | The job with job number *job_number*. |
| *%%* | The current job. |
| *%+* | The current job. |
| *%-* | The previous job. |

# lbpcat

**Purpose:** To echo load balancer packets.

**Format:** *<component> [n]||| lbpcat*

## Description:

**lbpcat** is a simple utility that just echoes load balancer packets. It can be used as a test worker-task which simply echoes all packets it is sent. For example,

```
controller [20]||| lbpcat
```

tests that the controller and load balancer are creating and routing packets correctly.

# lcontrol

**Purpose:** To convert stream mode to line mode.

**Format:** *<component> lcontrol < > lb ....*

## Description:

The **lcontrol** utility is a simple front end for the Helios load balancer. As described in *The Helios Operating System*, Section 7.5, the load balancer uses a specific packet protocol for all communication with a control process. Because you can change the protocol of the streams that connect the load balancer to the worker tasks into line mode, so that you can use standard text processing utilities as worker processes, you need to be able to convert a stream in line mode to a packet stream for input to the load balancer; **lcontrol** enables you to do this. So, to create a distributed concatenator, you could give:

```
cat <filename> | lcontrol [4]|| mycat
```

In this example, **lcontrol** is used only to set the load balancer into line mode and to pass on the lines output by the controlling **cat**.

# map

**Purpose:** To display activity in a Helios node.

**Format:** *map*

## Description:

The **map** command displays a help page, listing valid commands and a summary of the map format. Each of these commands is selected by a single key press, as shown below:

| Key | Action |
|---|---|
| q Q ESC | Terminate **map**. |
| h H | Display help page. |
| - | Halve sample rate. |
| + | Double sample rate. |
| *any other key* | Resize display and redraw. |

The map consists of four fields. The top row displays the total amount of memory which has been allocated, the amount which is free, and the number of bytes which each character in the map represents. Along the right hand side of the display is a table showing a list of active tasks and a letter which has been assigned to that task; these letters are then used in the main map to represent each of the tasks. The main map occupies the centre of the screen; this shows the allocation of system heap. Each character in the map represents a number of bytes which is specified in the top row of the display. Character positions which are occupied by '.' represent free memory, '#' represents memory which has been allocated to the system, and '@' and '?' are used to represent unidentified allocations. All letters which appear in the map show the amount of memory which has been allocated to a task, and digits represent shared libraries. At the bottom of the display is a graph showing the current processor load; the horizontal bar at the end of the graph marks the maximum load, and '=' shows the current loading.

# netversn

**Purpose:** To provide licencing details for the given Network Server.

**Format:** *netversn <subnet_name>*

## Description:

**netversn** gives licencing information about the Network Server specified as *subnet_name*. It displays the information about the type of licence which was issued (Single Machine or Network), the distributor's identification code, and the server's serial number.

# reset

**Purpose:** To reset the given subnetwork.

**Format:** *reset <subnet_name>*

## Description:

**reset** resets the given subnetwork. The main restriction to the application of the **reset** command is that it must not be used to reset the whole subnetwork, as this would destroy the root Network Server and put the network into an unrecoverable state. If you wish to reset the whole network, you should re-boot Helios.

# run

**Purpose:** To run a command in its own window.

**Format:** *run <command>*

## Description:

The **run** command creates a window, executes the specified command, and closes the window when the command terminates. The specified command is located by the path environment variable.

# runb

**Purpose:** To submit a job description to the Batch Server.

**Format:** *runb <file>*

## Description:

The **runb** command submits the specified job description file, ***file***, to the Batch Server, and then returns to the caller. A full description of the file's syntax is given in Section 3.3, "Batch Server".

# startns

**Purpose:** To start a local Network Server.

**Format:** *startns [options] <subnet_name>*

## Description:

**startns** is used to startup a local Network Server. This command can be entered into the shell, or used in the startup file, *initrc*. The options are passed to the Network Server and are as shown below:

| Option | Action |
|---|---|
| -r | Reset everything. |
| -nr | No reset; on booting, do not attempt automatic reset. |
| -nt | Do not create Task Force Manager for this network. |
| -nb | Do not boot this network. |

The above options may also be combined; for example, *-nrnt* specifies that the network is not to be reset and that no Task Force Manager is to be created (a Task Force Manager is created by default).

# tcp

**Purpose:** To copy files, converting CR/LF to LF.

**Format:** *tcp <filename> <filename>*
*tcp <filename> [<filename>] ... <dir>*

## Description:

**tcp** copies one or more text files. It is similar to **cp**, except that when **tcp** copies a file it also translates CR/LF to LF (that is, carriage return/linefeed to linefeed). This translation is necessary when you copy text files from an external filing system such as MS-DOS, which stores end of line as CR/LF, into an internal filing system such as the Helios filing system or RAM disc, which uses just LF. See *The Helios Operating System* manual for details on **xlatecr**.

## 3.8 Network Support

In order to run networked Helios, you will need to have two different copies of Helios booted from two separate hosts. Each host should be responsible for one subnetwork; at the minimum this means one transputer each. The resource maps used in both machines must correspond to the subnetwork to be booted by the Network Server in that subnetwork and must be given a unique subnet name. Any program requested to be run by the TFM within either subnetwork will not use any processor in the other subnetwork.

Each subnet description must describe any **external links**; these are the links which are used to cross connect the two subnetworks. External links are identified by number, normally starting at 0. The syntax used is that, instead of specifying a terminal component name such as ~01, you must specify ext*[n]* where *n* is the external link number. Each end of the external link must be defined, one end in each subnetwork. An example should help clarify this.

```
subnet /ClusterA {
        CONTROL Rst_Anl [/ClusterA/00];
                terminal 00 { ~IO, ext[0]  ,  ,;
                                        HELIOS;
                                        Mnode   Rst_Anl [im_ra_b4.d];
                                        ptype T414; }
                terminal IO { ; IO;}
}
```

Here the first subnet ClusterA has one transputer. Link 1 is designated the initial external link. This resource map is compiled and a Network Server started in order to boot this and run the associated TFM. In this case, the Network Server should be started by **startns**; for example,

```
startns /helios/etc/ClusterA.map
```

The following example defines the map for ClusterB, which shows the external link connected to link 3 of the single transputer in this subnet. The other copy of Helios should be booted and run with the map provided for the Network Server and TFM. Once again, the Network Server should be started by **startns**.

```
subnet /ClusterB {
        CONTROL Rst_Anl [/ClusterB/00];
                terminal 00 { ~IO,  ,  , ext[0];
                                       HELIOS;
                                       Mnode   Rst_Anl [im_ra_b4.d];
                                       ptype T414; }
                terminal IO { ; IO;}
}
```

In order to cause the network to become connected, a further version of the Network Server must be run in one transputer somewhere on the network. This must be started by **startnet**; the optional flags, *-nt*, should be used if users do not wish to share processors. This network-wide Network Server must be provided with a network map, which identifies the ways in which the subnetworks are interconnected via the external links. In our simple example, the map to be used would look as follows:

```
subnet /Net {
                subnet ClusterA { /Net/ClusterB; HELIOS; }
                subnet ClusterB { /Net/ClusterA; HELIOS; }
}
```

The initial external link is used as the first item in the external connection list. Any further external links could be added as subsequent items, with each item in the list separated by a comma. The subnetwork addresses in the link definitions must include the full pathname of the target subnetwork; '~' is not valid at this level.

### 3.8.1 Control System

The network control system is provided by a distributed Helios server called the Network Server (NS). This service is responsible for booting the network and for its subsequent control. The network must first be defined by the use of a text file called a **resource map**; the format of which is described in *The Helios Operating System* (Prentice Hall, 1989). The resource map is read by the NS at system boot time and is used to boot the defined network. Once the network is booted, the NS automatically installs the Task Force Manager hierarchy which enables the automatic allocation of programs (task forces) to processors.

The network control commands provide a simple command line interface to a resident library called the Network Control Library (*net_ctrl*). This is a library of routines which send requests direct to the relevant NS to perform the control functions. For example, the processor /net/machine1/04 can be

reset by the following command line:

```
reset /net/machine1/04
```

Assuming that your hardware supports individual reset, the **reset** command calls the function **Reset()** in *net_ctrl*, which sends a reset request to the Network Server responsible for this processor (/net/machine1/ns). Similarly, it is possible to reset a whole subnetwork. For example,

```
reset /net/machine1
```

will, if possible, reset all the processors in subnetwork /net/machine1.

### 3.8.2 Network Commands

This section describes the set of network control commands distributed with the network toolkit. These commands are exclusive to the networking Helios system and provide a command line interface to the network control system.

# clnames

**Purpose:** To clear all the name tables in a subnetwork.

**Format:** *clnames <subnet_name>*

## Description:

**clnames** clears all the name tables in the given subnetwork. Helios provides a distributed name service, with each processor maintaining its own table of object addresses (see *The Helios Operating System*, Section 15.2). It is desirable to flush these name tables under certain circumstances in order to force a new search for a given object. Entries in the name table for objects local to the processor (for example, a local server) are not removed.

## Example:

```
clnames /net/clustera
clnames /net/clustera/04
```

# connect

**Purpose:** To connect two subnetworks.

**Format:** *connect <subnet_A_name> <subnet_B_name>*

## Description:

**connect** connects two subnetworks. It must be possible to access the Network Server (NS) responsible for *<subnet_A_name>* from the processor on which this request is issued. So the request

```
connect /net/clustera /net/clusterb
```

will not succeed if initiated from subnetwork */net/clusterb*, but the following should work:

```
connect /net/clusterb /net/clustera
```

# cupdate

**Purpose:** To update a network context.

**Format:** *cupdate <subnet_name>*

## Description:

**cupdate** updates the network context of the given network. This command is used to update a partial network context to the full network context, and may be used when re-connecting errant subnetworks. For example,

```
cupdate /clustera /net/clustera
```

will update the network address (context) of every processor in the subnet *clustera*, and update the context of the required Network Servers and Task Force Managers.

# dconnect

**Purpose:** To disconnect two subnetworks.

**Format:** *dconnect <subnet_A_name> <subnet_B_name>*

## Description:

**dconnect** disconnects two subnetworks. For example:

```
dconnect /net/clustera /net/clusterb
```

# lrecon

**Purpose:** To reconfigure local processor link status.

**Format:** *lrecon*
*<link0_mode> <link1_mode> <link2_mode> <link3_mode>*

## Description:

The **lrecon** utility does not use the network control library at all. It calls the kernel directly to change the state of the local processor links, using the kernel **Reconfigure**. It is useful for changing link modes from the command line. For example,

```
lrecon 2112
```

will disable links 1 and 2.

**Note:** *<link_mode>* = 1 or 2 ( dumb or intelligent).

# lstatus

**Purpose:** To return the status of a network link.

**Format:** *lstatus <subnet_name> <link_number>*

## Description:

**lstatus** returns the status of the given network link. The status is returned in the format of the LinkConf structure (see *link.h* and *config.h*). The State and Mode are most useful as these define the current status of the link. For example,

```
lstatus /net/clustera/04 0
```

will return the status of this physical link.

# native

**Purpose:** To return a subnetwork to the native state.

**Format:** *native <subnet_name>*

## Description:

**native** reverts the given subnetwork to the native state. This involves terminating the network control servers (NS and TFM) in this subnetwork and resetting the member processors. It is not possible to revert the whole network to native in this way (see **boot**). This will also disable any links which disconnect this subnetwork (that is, any links which connect this subnetwork to the still active outside world). For example:

```
native /net/clustera
```

**Warning: native** should be used with care as it will kill any programs running in the target subnetwork. It does not currently terminate user programs.

# sfnc

**Purpose:** To update the system function entry in the NS and TFM database.

**Format:** *sfnc <subnet_name> <system_function>*

## Description:

The **sfnc** command updates the system function entry in the distributed database of the NS and TFM. The system function defines what a subnetwork may be used for. Only processors with the function HELIOS will be considered for running user programs. If you wish to prohibit further placement of programs in a particular processor, you can invoke this command with the **SYSTEM** function:

| Option | Description |
|---|---|
| NATIVE = 1 | (You should use the **native** command not this option) |
| HELIOS = 2 | This is a HELIOS subnetwork; can load user programs |
| IO = 3 | This is an IO Subnetwork |
| SYSTEM = 4 | This is a SYSTEM subnetwork; cannot load user programs |

## Example:

```
sfnc /net/clustera/03 4
```

# smemory

**Purpose:** To update the memory entry in the NS and TFM database.

**Format:** *smemory <subnet_name> <memory_size>*

## Description:

The **smemory** command updates the memory entry in the distributed database of the NS and TFM. The memory field defines the total amount of memory available in the given subnetwork, *<subnet_name>*, (that is, *<memory_size>* = new memory size for the subnetwork in bytes), and is used in the mapping of task forces, where memory requirements have been specified in the CDL definition. For example:

```
smemory /net/clustera 2000000
```

# sptype

**Purpose:** To update the processor types in the NS and TFM database.

**Format:** *sptype <subnet_name> <processor_type> <number_processors>*

## Description:

The **sptype** command updates the processor types entry in the distributed database of the NS and TFM. The processor types field defines the total number of processors of each type in the given subnetwork.

**Note:** *<processor_type>* = 2 or 3 or 4 (T414, T800 or 68000); *<number_processors>* = number of processors of given type.

# sstatus

**Purpose:** To return the Network Server status for a subnetwork.

**Format:** *sstatus <subnet_name>*

## Description:

**sstatus** returns the internal NS state for the given subnetwork; it is really only of any use when trying to determine why a subnetwork has not been booted. Valid Network Server states include:

| State | Description |
|---|---|
| ACTIVE = 1 | The subnetwork is active (fully booted) |
| PENDING = 2 | The subnetwork is changing to ACTIVE (partially booted) |
| UNKNOWN = 8 | State unknown (still trying to determine state) |
| UPDATE = 16 | The NS is updating its database |

# startnet

**Purpose:** To start up a network-wide Network Server.

**Format:** *startnet [options] <map_name>*

## Description:

The **startnet** command starts up a network-wide Network Server. It should be used after **startns** has been used to start up the local Network Servers. Valid options for the Network Server are as follows:

| Option | Action |
|---|---|
| -r | Reset everything. |
| -nr | Do not attempt automatic reset on booting (set by default). |
| -nt | Do not create Task Force Manager for this network. |
| -nb | Do not boot this network. |

The above arguments may also be combined; for example, *-nrnt* specifies that the network is not to be reset and that no TFM is to be created.

By specifying the option *nt*, you can share the available processors with other users. In general, this is inadvisable as there is currently nothing to stop a user from resetting another user's processor.

## 3.9 The Session Manager

The Session Manager (SM) is a system service responsible for creating user sessions and for restricting the total number of active sessions in a Helios system. It reads in the standard Helios password file, *etc/passwd*, and generates an internal database of users, */sm/userdata*. The list of potential users can thus be read from the SM by opening the directory */sm/userdata* and then reading it. Although the SM does not currently support the reading of the userdata entries themselves, future versions will allow this.

The SM services requests to create a session. When the SM is requested to create a session it expects to be given an initialised SessionInfo data structure which identifies the user. Having validated the user name and password, the SM creates a session entry and returns the created session; this represents a single user's initialised session data. Several Open requests may then be made to create session environments, each one generating a RequestEnv request to produce the environment. The resulting environment, which is created for each session, includes the root program name (usually a shell), the user's home directory, the user's console and keyboard streams, and other attributes. A record of each active session is maintained in the directory */sm/users*. A list of the users who are currently logged into the Helios network can then be obtained by listing the contents of this directory.

The interface to the SM is usually provided by a login worker process called **smlogin**. This is equivalent to the basic login program but it uses the SM to verify the username and password and to create the login shell environment as described above.

The password file may be updated to change the parameters or the number of valid users. In the present implementation it is not possible for users to change their own passwords; to do this, the system administrator must change the password file directly.

```
UserName:Passwd:GID:UID:Comment:HomeDirectory:RootProgram [args]

GID = Group Id
UID = User Id
Comment = a text comment (usually full name of user)
HomeDirectory = the users home directory
RootProgram = the root program (usually /helios/bin/shell)
```

The **smlogin** utility is a login service. It displays the login prompt, creates a session (within the Session Manager), and sets up a login shell. When the login shell is exited, **smlogin** will re-display the login prompt to allow another user to log in. **smlogin** will also allow a user to log in through a remote console if it is provided with the stream which will be opened for the login console. For example, to log in through a remote Window Server, you could type something like this:

```
smlogin -d /fred/00/window/tty2 &
```

which would create a login worker for the stream */fred/00/window/tty2*.

## 3.10 New Functions

The following functions have been added in to Helios 1.1:

*word *InitProcess(word *stack, VoidFnPtr entry, VoidFnPtr exit, word *display, word nargs);*

**InitProcess** initialises a process for execution. *Stack* points to the top of the memory to be used as the stack, *entry* is the code to be executed in the process and *exit* the return address for when this returns. *Display* points to the initialised display passed to the initial call. *Nargs* is the number of bytes of arguments to be passed in. The stack is initialised according to the standard calling conventions and a pointer returned to the space left for the arguments.

*void StartProcess(word *p, word pri);*

**StartProcess** starts a process initialised by **InitProcess** at the given priority.

*void StopProcess(void);*

**StopProcess** halts the current process.

*word GetPortInfo(Port port, PortInfo *info);*

**GetPortInfo** fills the provided PortInfo structure with information about the port.

*void FreeMemStop(void *mem);*

**FreeMemStop** frees the memory block and halts the current process. This function is used to allow the stack on which the current process is executing to be released without it being re-allocated before the process has a chance to stop itself.

*void SignalStop(Semaphore *sem);*

**SignalStop** signals the semaphore and halts the current process. Like **FreeMemStop**, this function is used to prevent problems in memory allocation.

*word Configure(LinkConf newconf);*

This function is used to re-configure a single processor link. **Configure()** should be used instead of **Reconfigure()**.

*Stream *PseudoStream(Object *object, word mode);*

**PseudoStream** manufactures a stream of Type_Pseudo to the given object. Unlike normal streams this will not be opened. It may be used wherever a normal stream can be used, and will be opened automatically if necessary.

## 3.11 Changed Functions

The following function, which exists in Helios 1.0, has changed in specification in Helios 1.1:

*PUBLIC word GetEnv(Port port, Environ *env);*

The Port argument has been added to **GetEnv**, to allow environments to be passed to any port.

## 3.12 Extended Functions

The specification of the following function has been extended in Helios 1.1:

*void *Malloc(word size);*
if size = -3 result is total size of heap.

# Index


^^ 3.16
^^ constructor 3.15
| constructor 3.15, 3.16
||| constructor 3.15
< 3.14
< > constructor 3.15, 3.17
~ (tilde) 3.4
! 3.7
!! 3.7
# 1.6, 2.5, 3.7
0x 3.7
% 3.1, 3.15
%1 (termcap) 2.6
@7 (termcap) 2.6
&8 (termcap) 2.6
& 3.1, 3.2
/alias - see also Alias Server 3.2
/bin 1.2 - see also /helios/bin
/etc 1.2 - see also /helios/etc
/etc/hosts 1.5
/etc/services 1.5
  - see also System configuration file
/fifo 3.8
/files 2.7
/helios 2.7
/helios/etc 2.8
/helios/etc/initrc 2.8
/helios/lib 1.6
/helios/lib/init 1.6
/helios/lib/window 2.2
/include 1.2
/lib 1.2 - see also /helios/lib
/logger 1.6, 1.7, 2.8, 2.9

auto 1.6
Automatic allocation of streams 3.16
Auxiliary
  - list stream allocation 3.18
  - streams 3.18
Available memory 3.50

Batch Server 1.6, 1.8, 3.3, 3.6,
  3.36 - see also runb
Bell sequence 2.7
bin 3.12
bl (termcap) 2.7
boot 3.22, 3.23, 3.48
Boot (debugging option) 2.3
Booting - see also boot
  large subnetworks 3.23
  network 3.39
  processors 1.4
  subnetworks 3.23, 3.52
bootlink 2.14
BootLink() 3.8
Bootstrap 2.13
Busy system 2.12

c 3.22, 3.24, 3.25
C compiler optimisation 3.8, 3.9
C library 3.8
CDL 3.5, 3.9-3.22, 3.50
  - code (component attribute) 3.9
  - compilation errors 3.19
  - compiler 3.10, 3.12, 3.19
  - new syntax 3.21, 3.22
  - scripts 3.19, 3.20

ce (termcap) 2.7
cl (termcap) 2.7
Clear screen 2.7
Clear to end of line 2.7
clnames 3.42
Close (debugging option) 2.3
cm (termcap) 2.7
co (termcap) 2.7
code (component attribute) 3.9
  - see also CDL
Command name subscript
  expressions 3.15
Commands, running - see run
Comments 1.6
Communication between processes 3.8
Compiler driver - see c
Compiling and linking a program
  - see c
Component Distribution Language
  - see CDL
Component attributes 3.9
  - see also CDL
Component declaration - see CDL
Configuration file 1.3, 1.4, 1.5,
  1.7, 2.1, 2.2, 2.4, 2.5, 2.13,
  2.14, 2.15
  - see also host.con, hydra.con
  - re-read (<hot key>-z) 2.5
  - re-read - see Reconfigure
  - specify - see server
Configuraton, network 1.5
Configure 3.56
connect 3.43
Connecting subnetworks - see connect
Connection problems 2.12
console 1.6, 1.7
Console window 1.7
Constructor(s) 3.11-3.18
  - see also CDL
  - <> 3.17
  - ^^ 3.16
  - | 3.16
  - farm 3.12
  - interleave 3.12, 3.13, 3.14
  - pipe 3.18
  - precedence 3.15, 3.16, 3.17
  - subordinate 3.18
Context update, network - see cupdate
Converting stream mode to line mode
  - see lcontrol
copy 3.0
Copying files 3.38
  - see also cp, tcp
CR/LF to LF translation 3.38
  - see also tcp
Crashing the transputer 2.13
Creating a new session 3.54
CTRL-G - see Bell sequence
cupdate 3.44
Current process, stopping 3.55, 3.56
Current window refresh (<hot key>-3) 2.5
Cursor move 2.7

dconnect 3.45
Debugger 2.2-2.5
  - enter (<hot key>-7) 2.5
debugger_key (function key k5) 2.5
Debugging 2.2-2.5
  - facilities menu 2.2, 2.3
  - options 2.3, 2.5
  - resources (<hot key>-x) 2.5
  - window 2.2
Device servers, progress report
  while starting - see Init
Directory structure mapping 3.2
Disable link 3.26, 3.46, 3.48
  - see also dlink
Disconnecting a server 2.12
Disconnecting subnetworks
  - see dconnect
Display Helios node activity
  - see map
Distributed searches, report
  - see Search
dlink 3.22, 3.26
Down-arrow key 2.6
Dumb mode links 3.26
Dumb terminals 2.4, 2.15

Echo load balancer packets
  - see lbpcat
elink 3.22, 3.27
Enable link 3.27
  - see also elink
End key 2.6
Enter debugger (<hot key>-7) 2.5
Environment - pass to port 3.56
Environment of job 3.5
Error
  - codes 3.6
  - log 2.8, 2.9 - see also /logger
  - logger 1.7, 2.2, 2.4, 2.5,
  2.8, 2.9
  - messages from the Server 2.4
  - output destinaton 2.8, 2.9
    - see also /logger
Escape keys 2.5
Escape sequences 2.6
etc/batchrc 1.6, 1.8
etc/initrc 1.6

Ethernet 2.9, 2.10
Executable task force, specify 3.5
exit_key (function key k7) 2.5
Exit Server (<hot key>-9) 2.5
External links 3.39, 2.40

Farm constructor 3.12
  - see also Interleave constructor
Fault 3.6
  - codes 3.6
  - database 3.6, 3.7
  - database, private 3.7
Fault library 3.6
Fault() 3.6
fdbclose 3.6, 3.7
fdbfind 3.6, 3.7
fdbopen 3.6, 3.7
fdbrewind 3.6, 3.7
fg 3.1, 3.22, 3.28
Fifos 2.8, 3.8
File(s)
  - being closed, list - see Close
  - being opened, list - see Open
  - copying 3.38 - see also cp, tcp
  - descriptors 3.18
  - reads, report - see Read
  - writes, report all - see Write
Filing system interface 2.7
filter 3.11
Floating point 3.9
font 1 3.0
Foreground job - see fg
FreeMemStop 3.56
Freeing memory blocks 3.56
Function
  - code format 3.8
  - key operations 2.5, 2.6

GetEnv 3.56
gethostbyname() 2.11
GetPortInfo 3.55
getservbyname() 2.11
Give names of objects being accessed
  - see Name
Graphics (debugging option) 2.3
Graphics operations, report any
  - see Graphics

Halting the current process 3.55, 3.56
helios 1.1, 1.2, 1.3
  - directory/ies 1.2, 1.3
  - helios/bin 1.2
  - helios/etc 1.2, 1.5
  - helios/etc/initrc 1.4
  - helios/include 1.2
  - helios/lib 1.2
  - helios/tmp 1.2
Helios 1.1, new commands in 3.22
Helios directory/ies - see helios
Helios node, display activity in
  - see map
Help key 2.6
Home key 2.6
host.con 1.3-1.5, 2.1, 2.2, 2.4,
  2.5, 2.7, 2.9-2.15
  - example file 2.15
  - see also Configuration file
Hot keys 2.1, 2.4, 2.5, 2.6, 2.8
Hydra link daemon 1.2, 1.3, 1.5, 2.10,
  2.11, 2.12 2.13, 2.15
  - overloading 2.13
  - recovery 2.13
hydra.con 1.5, 2.12, 2.13
  - see also Configuration file
hydramon 1.2, 1.5, 2.12

ifabsent 1.6
io_processor 2.14
I/O Server 1.1, 1.2, 1.3, 1.6, 1.7,
  2.1-2.15
  - special key sequences 2.4
  - window 2.4
  - processor 1.6, 2.14
  - processor connection 2.14
ITFTP32 2.10
init 1.7
Init (debugging option) 2.3
InitProcess 3.55
Initialisation file
  (/helios/etc/initrc) 2.8
Initialised process, starting an 3.55
Initialising a process 3.55
initrc 1.6
Insert key 2.6
Installation 1.1-1.8, 2.11, 2.12
  - administration 2.11, 2.12
Intelligent mode links 3.27
Inter-task communication 3.8
Interleave constructor(s) 3.12, 3.13,
  3.14
Internet 1.5, 2.11
internet 2.13
Inverse video 2.7
Iteration names 3.14, 3.15

jobs 3.1
Job(s) 3.1, 3.2, 3.3, 3.5, 3.6
  - see also fg, jobs, Task force
  - bring into foreground 3.28
  - control 3.1, 3.2

- creation 3.6
- description file 1.8, 3.6, 3.36
  - see also etc/batchrc
- environment 3.5
- list 3.1, 3.2
- numbers 3.1
- objects, specifying 3.5
- priority 3.4
- remote execution 3.3
- repeat delay 3.4
- rescheduling 3.4
- start time 3.3, 3.4
- status 3.4
- terminate - see kill

k1-k9 (termcap) 2.6
k; (termcap) 2.6
kd (termcap) 2.6
Kernel 3.8, 3.46
Key presses, report all - see Keyboard
Key translation 2.6 - see also Termcap
Keyboard (debugging option) 2.3
kh (termcap) 2.6
kI (termcap) 2.6
kill 3.1, 3.22, 3.29
Killing programs 3.48 - see also kill
kl (termcap) 2.6
kN (termcap) 2.6
kP (termcap) 2.6
kr (termcap) 2.6
ku (termcap) 2.6

Large subnetworks, booting 3.23
lb 3.12
lbpcat 3.22, 3.30
lcontrol 3.22, 3.31
Left-arrow key 2.6
Licencing 3.33
Link
- adapter 2.9, 2.15
  - multiple 2.9
- daemon - see Hydra
- definitions 3.40
- disable 3.26 - see also dlink
- dumb mode 3.26
- enable 3.27 - see also elink
- intelligent mode 3.27
- modes, changing 3.46
- non-blocking mode 2.13
- status 3.47

LinkConf 3.47
Linking a program - see c
Links, reconfiguring 3.46
List files being closed - see Close
List files being opened - see Open
Listing open streams - see Resources
Load balancer/ing 3.12, 3.30, 3.31
- packet protocol 3.31
- packets, echo - see lbpcat
- see also lb 3.12

Loading server 1.7
Local Network Server 3.37, 3.54
- start - see startns

Local processor links, reconfigure
- see lrecon

Locking sites 2.13
Logfile 2.9
logfile 2.9
Logger device 2.8
login 1.6
Login 3.54
Low level routines 3.7
lrecon 3.46
lstatus 3.47

Main Helios directory location, specifying the 2.7
Malloc 3.57
map 3.22, 3.32
Mapping directory structure 3.2
Master/slave 3.18
me (termcap) 2.7
Memory
- allocation 3.56
- available 2.14, 3.50
- blocks, free 3.56
- specifying the amount of 2.14

message_limit 2.14
Messages
- (debugging option) 2.3
- maximum size of 2.14
- passing 3.8

Monitoring program - see hydramon
mr (termcap) 2.7
mul 3.8
Multi-dimensioned replication 3.13
Multiple
- copies of Helios 2.8
- link adapter 2.9
- links 2.9
- users 2.9, 2.12, 2.13
- windows 2.1, 2.2

Name 2.3
- (debugging option) 2.3
- table 3.8, 3.42
  - clear - see clnames

Name Server 1.6, 1.7
native 3.48, 3.49
Native state - see native

net_ctrl 3.40, 3.41
- see also Network control library
netversn 3.22, 3.33
Network 2.14
- address (context) 2.13, 3.44
- booting 3.39, 3.40
- commands 3.41
- configuration 1.5
- context update - see cupdate
- control 3.40, 3.41, 3.46, 3.48, 3.49, 3.50
  - see also Network Server, TFM
  - library 3.40, 3.41, 3.46
  - servers, terminating 3.48
  - system 3.40, 3.41
- licence - see also netversn 3.33
- link status 3.47 - see also lstatus
- map(s) 2.8, 3.40
- messages, report on - see Messages
- resetting the 3.34
Network Server 1.4, 3.23, 3.39, 3.40, 3.43, 3.44, 3.48
- see also NS
- licence 3.33
- start - see startnet, startns
- status - see sstatus
Networked Helios 3.39
Networking 1.3, 1.4
New Helios functions 3.55
New commands in Helios 1.1 3.22
New connection problems 2.12
Next window, switch to (<hot key>-1) 2.5
Node activity, display - see map
Non-blocking mode 2.13
Normal video 2.7
NS 3.40, 3.43, 3.48, 3.49, 3.50, 3.51, 3.52
- see also Network Server
- distributed database 3.49, 3.50, 3.51

Object address tables
- see Name tables
Objects being accessed, give names of - see Name
Objects passed in job environment, specify 3.5
Open 3.54
- (debugging option) 2.3
- files 3.8
- streams 2.3, 3.5, 3.9
  - see also Resources
Overloading Hydra 2.13

PageDown key 2.6
PageUp key 2.6
Parallel constructor ^^ 3.16
Password file 3.54
pipe 3.11
Pipe(s) 3.8, 3.11, 3.12, 3.13
- constructor | 3.16, 3.18
Pipe Server 3.8 - see also /pipe
Port
- information 3.55
- passing an environment to 3.56
- table garbage collector 3.8
Porting
- makefiles 3.2
- shell scripts 3.2
Precedence of constructors 3.15
Previous window, switch to (<hot key>-2) 2.5
Priority of jobs 3.4
Private fault database 3.7
Process
- identification 3.1
- initialisation 3.55
- starting a 3.55
- stopping 3.56
Processing initialisation 3.55
Processor(s)
- manager - see Processor Manager
- names 2.14
- performance monitor 3.8
- type 3.51
- sharing 3.40, 3.53
Processor Manager 3.8
prod 3.8
Program scheduling 3.3
Program to processor allocation 3.40
Programs, remote execution of 3.3
Progress report(s) 2.3
- during transputer bootstrap
  - see Boot
- while Server is exiting
  - see Quit
- while device servers are starting - see Init
Protocol 2.12
Pseudo windows 2.1
PseudoStream 3.56

Quit (debugging option) 2.3

Read (debugging option) 2.3
Real windows 2.1, 2.2
Reboot transputer (<hot key>-0) 2.5
Rebooting 2.4
reboot_key (function key k8) 2.5

Reconfigure 2.3, 3.46, 3.56
- (debugging option) 2.3
- local processor links 3.46
- single processor link 3.56
Reconnecting subnetworks 3.44
Refresh current window
(<hot key>-3) 2.5
refresh_key (function key k4) 2.5
Remote
- access 1.5
- execution of programs 3.3
- login 3.55
Repeat delay, job 3.4
Repeating jobs 3.4
Replication 3.11, 3.13, 3.14
- limit 3.14
- multi-dimensioned 3.13, 3.14
Replicators 3.11
- see also CDL
Report all file reads - see Read
Report all file writes - see Write
Report all key presses
- see Keyboard
Report any graphics operations
- see Graphics
Report distributed searches
- see Search
Report on network messages
- see Messages
RequestEnv 3.54
Re-read configuration file 2.3, 2.5
- see also Reconfigure
Rescheduling jobs 3.4
reset 3.22, 3.34, 3.41
Reset driver - see tram_ra.d
Reset() 3.41
Resetting a subnetwork 3.34, 3.41
- see also reset
Resetting network processors 3.48
- see also native
Resetting the network 3.34
Resident libraries 3.40
Resizing windows 2.2
Resource
- debugging 2.3, 2.5
- map(s) 1.4, 3.39, 3.40
Right-arrow key 2.6
ro (termcap) 2.7
Root Network Server 3.34
root_processor 2.14
Root transputer 2.9
run 1.6, 1.7, 1.8, 3.2, 3.22, 3.35
runb 3.6, 3.22, 3.36
Running a command - see run
Screen operations, translating
escape sequences to 2.6
Screen size 2.2
se (termcap) 2.7
Search (debugging option) 2.3
server 1.3. 2.1, 2.3
Server 1.5, 2.1-2.15
- see also I/O Server
- configuration file 1.3
- disconnecting a 2.12
- exit (<hot key>-9) 2.5
- exiting, progress report while
- see Quit
- loading 1.7
- status (<hot key>-8) 2.5
serverwindow 1.2, 2.3
serverwindow.sun3 2.3
Session(s)
- active 3.54
- creation 3.54
- environment 3.54
- user 3.54
SessionInfo 3.54
Session Manager (SM) 3.54
sfnc 3.49
Sharing processors 3.40, 3.53
Shell 3.1, 3.2, 3.8
SignalStop 3.56
Single machine licence 3.33
- see also netversn
Site(s) 1.1, 1.2, 1.4, 1.5, 2.9,
2.10, 2.11, 2.13
- see also Transputer site
- access 1.1, 2.13
- administration 1.2, 2.11
- allocation 1.3, 1.4, 1.5
- locking 2.13
- numbers 1.1
- release 2.12
- unused 1.5
SM 3.54 - see also Session Manager
smemory 3.50
smlogin 3.54, 3.55
so (termcap) 2.7
Socket(s) 2.8, 2.11, 2.12, 2.13
- identifier 2.11
- internet 2.11, 2.13
- TCP/IP 2.10, 2.11
- Unix 2.11, 2.13
Special keys 2.2, 2.6
sptype 3.51
sstatus 3.52
Start(ing)
- Helios 1.6
- initialised process 3.55

- local Network Server - see startns
- time, job 3.3, 3.4
startnet 3.40, 3.53
startns 1.8, 3.22, 3.37, 3.39, 3.53
StartProcess 3.55
Startup file 3.37
Startup options 1.6
status_key (function key k6) 2.5
Status
- job(s) 3.4
- Server (<hot key>-8) 2.5
StopProcess 3.55
Stopping the current process 3.55, 3.56
Stream(s)
- allocation 3.16, 3.18
- mode to line mode conversion
  - see Icontrol
- passed as job environment, specify 3.5
Subnetwork(s) - see also Network(s)
- addresses 3.40
- booting 3.23, 3.52
  - see also boot
- connection 3.43
  - see also connect
- disconnection 3.45
  - see also dconnect
- memory available in 3.50
- processor types 3.51
- reconnecting 3.44
- resetting 3.34, 3.41
  - see also reset
Subordinate constructor < > 3.17, 3.18
Subscript expressions 3.18, 3.19
Subscripted component declarations 3.10
- see also CDL
Sun-3 1.1, 1.2, 2.3, 2.12
Sun-4 1.1, 1.2, 2.3, 2.12
sunbin 1.1, 1.2
Sun I/O Server 2.1, 2.2
SunOS 1.1
SunView 2.1, 2.2, 2.8
switch_backwards_key (function key k3) 2.5
switch_forwards_key (function key k2) 2.5
Switch error logger destination (<hot key>-l) 2.5, 2.8
Switch to next window (<hot key>-1) 2.5
Switch to previous window (<hot key>-2) 2.5
Symbolic links 2.8
System
- administration 2.13, 3.54
- configuration file 1.5
- function 3.49
- library 3.8
- services 3.5

Task force 3.3, 3.5, 3.10, 3.11, 3.16 3.19, 3.20, 3.40
- see also Job
- allocation 3.40
- configuration 3.16
- definition 3.10, 3.11, 3.19, 3.20
- executable 3.5
- object 3.5
- stream allocation 3.16
- use of streams 3.19
Task Force Manager 3.40, 3.44
- see also TFM
tcp 1.5, 3.22, 3.38
TCP/IP socket 2.10, 2.11
TDS 2.9
TERM environment variable 2.4
Termcap 2.4-2.7
- database 2.4
- entries 2.6, 2.7
- names 2.4
- sequence translation 2.6
Terminal
- input, dumb 2.4
- keys 2.6 - see also Termcap
- size 2.7
- wrapping characteristics 2.7
Terminate job - see kill 3.29
Text files, copying 3.38
- see also cp, tcp 3.38
TFM 3.8, 3.39, 3.40, 3.44, 3.48, 3.50, 3.51
- see also Task Force Manager
- distributed database 3.49, 3.50, 3.51
Tilde (~) 2.8, 3.4
Timeout errors 3.23
Toggle all debugging options (<hot key>-a) 2.5
tram_ra.d (Reset driver) 1.4
Transputer bootstrap, progress report during - see Boot
transputer_memory 2.14
Transputer site 1.1
- allocation 1.3

Undo key 2.6
Unix 2.1, 2.2, 2.4, 2.7, 2.9, 2.11, 2.13, 2.14
- filing system, accessing the 2.7, 2.8

Unused sites 1.5
Up-arrow key 2.6
Updating a network context
  - see cupdate
usadata - see User database
User changes to configuration 2.14
  - see also host.con
User database 3.54
User sessions 3.54
Users, list 3.54

waitfor 1.7
Waiting for the server to be
  loaded - see waitfor
Window(s) 2.1-2.4, 3.35
  - creation 2.2, 2.3
  - debugging 2.2
  - interface 2.1
  - I/O Server 2.4
  - multiple 2.1, 2.2
  - operations 2.2
  - pseudo 2.1
  - real 2.1, 2.2
  - resizing 2.2
  - switching 2.4
Window Manager 1.7
Window Server 1.7, 3.55
Workload balance
  - see Load balancing
Write (debugging option) 2.3