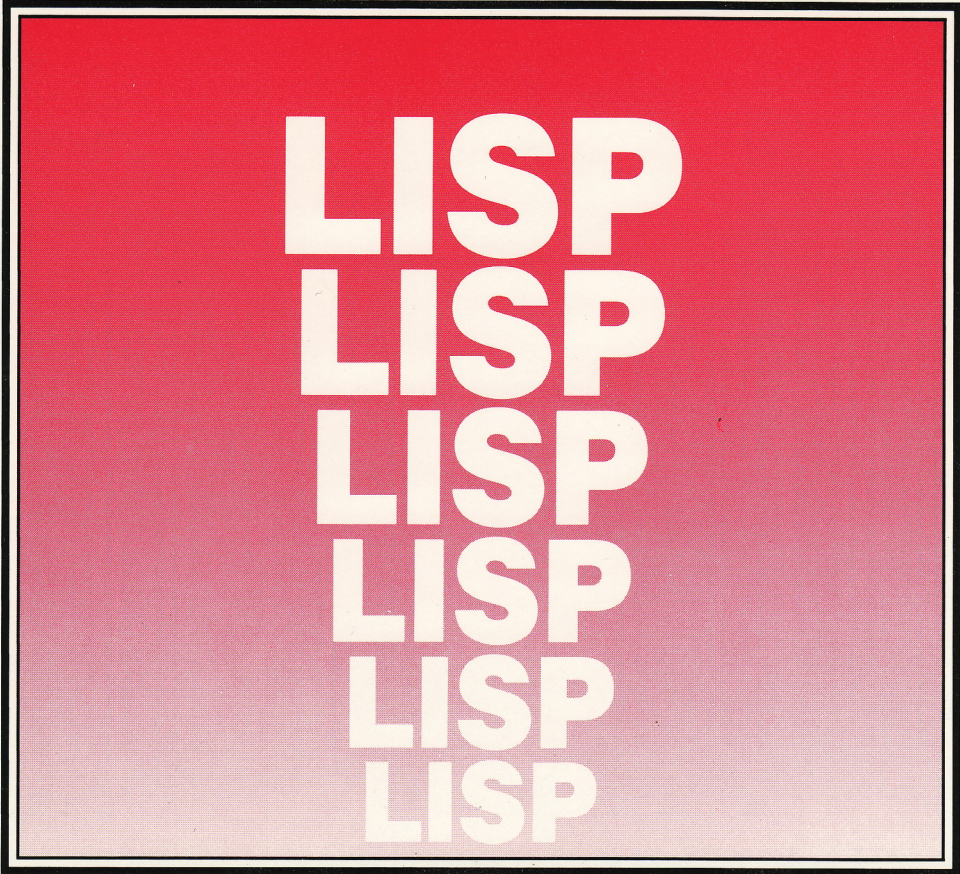


LISP Application Notes



LISP Application Notes

for HP 9000 Series 300 Computers

HP Part Number 98678-90010

© Copyright 1986 Hewlett-Packard Company

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Software clause in DAR 7-104.9(a).

Use of this manual and flexible disc(s) or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs can be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

© Copyright 1980, 1984, AT&T, Inc.

© Copyright 1979, 1980, 1983, The Regents of the University of California.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.

Hewlett-Packard Company

3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

May 1986...Edition 1. This manual documents release 1.0 of the common LISP Development Environment for HP 9000 Series 300 Computers.

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

WARRANTY

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Table of Contents



Chapter 1

Introduction

Purpose	1
Prerequisites	2
Topics	2

Chapter 2


Using a Large Heap

Introduction	5
Synopsis	5
Other Documentation	5
Background	6
Memory Layout	7
Windows/9000 Shared Memory	8
Swap Space	9
NMODE Resource Usage	9
Decisions	10
Swap Configuration	10
Process Size	11
Other Configuration Changes	12
Procedure	12
Overview	12
Making a File System	13
Creating a Configuration File	14
Running Config	16
After Booting	16

Chapter 3

The Example

Introduction	19
The Program	20
Overview	20
Instance Types	21
Windowing Utilities	22
Source Code	22



Chapter 4

Libraries

Introduction	37
Pointer Parameters	37
Constants	37
HP-UX System Calls	38
Special Considerations	38
Constants	40
Functions	40
Windows/9000 Functions	44
Constants	45
Functions	46
Starbase Graphics Functions	52
Constants	52
Functions	53
Device I/O Functions	60
Constants	60
Structure Arguments	60
Functions	61
Networking Functions	62

Chapter 5

User I/O

Introduction	63
Prerequisites	63
Organization	63
File I/O	64
Text Files	64
Terminal Device Files	65
Sequence of Events	66
Term0 Windows	71
Window-Smart and Window-Dumb	71
Window-Smart Example	71
Window-Dumb Example	72
Graphics Windows	73
Example	73
The Mouse	75
Pop-Up Menus	75
Example	75
Ideas for Expansion	78
NMODE I/O	79
Standard Input and Output Streams	79


NMODE I/O Functions	80
---------------------------	----

Chapter 6

Delivery

Introduction	81
Dump Files	81
Example	82
Drawbacks	82
General-Purpose Dump File	83
Using Command Line Arguments	83
Using a Script	85






Chapter 1

Introduction

Purpose

The purpose of the *Lisp Application Development Notes* is to provide a “cookbook” of techniques useful for developing applications on Hewlett-Packard’s Lisp workstation. For the most part, the topics covered are particular to HP hardware and software. They range from using the HP-UX software libraries to making a Lisp program accessible as an HP-UX command. In order to reinforce the concepts presented, an example program presented in the beginning of the manual is used where appropriate to illustrate the concept being discussed.



In the future, there may be a Common Lisp standard defined for window and human interface functions. Future releases of the Hewlett-Packard’s Lisp development environment may implement such a standard. We recommend that you isolate system-dependent code in application programs as much as possible so that porting to the standard (and thus to other machines) may be relatively painless. Since the current windowing system is just a Lisp veneer for HP-UX Windows/9000, any source code you write for it will be usable as long as Windows/9000 is supported by HP.

Remember that HP’s Common Lisp software is protected by a codeword and an HP46084A HP-HIL ID module. You must have purchased either 98679A (Execution License for Common Lisp) or 98678A (Development Environment for Common Lisp) for each runnable instance of your application. The execution license allows loading and executing any of the files listed in the file `$LISP/config/98679A-files`. See the *Installation and Overview* for more information about security modules and codewords.

Prerequisites

To use this manual effectively, you need experience with both Lisp and HP-UX. Some particular concepts mentioned here with little or no explanation are

- Calling non-Lisp routines
- Object-oriented programming
- HP-UX Windows/9000 concepts

You might want to have the following manuals handy when you are using this manual.

- *Lisp Programmer's Guide*
- *HP Windows/9000 Programmer's Manual*
- *HP Windows/9000 User's Manual*
- *HP-UX Reference*
- *Starbase Reference*

Topics

The following chapters comprise this manual:

- | | |
|---|--|
| Chapter 1
Introduction | The very same introduction you are now reading. |
| Chapter 2
Using a Large Heap | As it's shipped, your system cannot run a Lisp process much larger than eight megabytes. This chapter explains how to configure your system to run large Lisp processes. If the resources are available, this is desirable so that you can run applications that use large amounts of data, or to minimize the number of garbage collects. |
| Chapter 3
The Example | Presents the example program used throughout this manual. |
| Chapter 4
Libraries | Explains how to load and use the provided Lisp code for accessing HP-UX libraries such as Windows/9000 and Starbase graphics. |
| Chapter 5
User I/O | Explains how to get input from the keyboard and mouse, and how to output to windows. |
| Chapter 6
Delivery | Explains how to make a Lisp application "stand-alone", as well as make it executable from an HP-UX shell. |

NOTES






Chapter 2

Using a Large Heap

Introduction

For one reason or another, you may want to run a Lisp process with a larger heap than is possible with the standard configuration of HP-UX. If you're using Windows/9000 from your Lisp process (highly likely if you're running NMODE), then the standard configuration will allow a maximum Lisp process size of only around eight and a half megabytes, with about four megabytes allocated for dynamic heap (the other space is used for static heap). This default maximum size can be too restrictive for some applications.



Reconfiguring your system to run a larger Lisp process is not a trivial procedure. You will have to create a new HP-UX kernel with the *config(1M)* command, as well as install additional swap space (either on an existing disk or a new disk). This does not mean that you should be afraid to reconfigure, just that you should do it carefully.


Synopsis

This chapter contains the following sections.

Background	Provides information vital to understanding reconfiguration.
Decisions	Describes some choices you will have to make when deciding upon a new configuration.
Procedure	Describes the steps involved in reconfiguring your system.

Other Documentation

There are some other manuals that you should have handy when you are reconfiguring. These are from the documentation you received with your HP-UX system.

- The *HP-UX Reference*, particularly sections 1 and 1M.
 - The *HP-UX System Administrator Manual*. Chapter 5 and Appendix D contain the items of interest.
 - The *HP Windows/9000 Users's Manual*. See the "Resource Usage Considerations" appendix.
- 

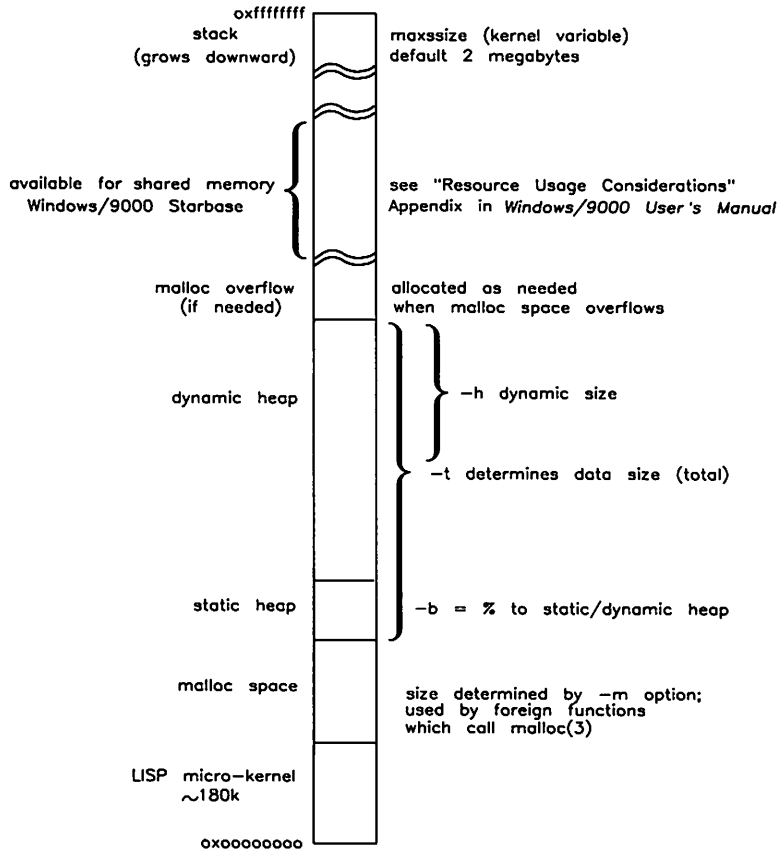
Background

Unfortunately, reconfiguring your system is not just a matter of deciding how big a heap you want to have. There are a few other things that need to be considered. This section identifies these things and describes a few of them.



Memory Layout

The following figure shows how the Lisp system's data structures are arranged in memory. This diagram is meant to serve only as a general guide. (The Series 300 Model 310 uses twenty-four bit addresses, so on that machine the execution stack grows downward from 0xFFFFF.)



Lisp Process Memory Map

Windows/9000 Shared Memory

Any HP-UX process that makes use of the windowing software shares a portion of its address space with the window manager process. Since most Lisp applications will want to use the windowing system, this shared memory becomes an important part of the memory map for the Lisp system. The shared memory is described in detail in the “Resource Usage Considerations” appendix of the *HP Windows/9000 User’s Manual*.

Note that just because a process runs in a window does not mean the process uses the window manager. For instance, if you bring up a Common Lisp read-eval-print loop in a window, it does not share memory with the window manager unless you make a call to one of the windowing functions. The default NMODE environment, on the other hand, makes extensive use of the windowing system, so its process does have the shared memory.

You need to be concerned with the shared memory because its default location limits a process’ text and data space to 8.75 megabytes. To run a larger Lisp process you will need to move the shared memory to a higher address. This entails changing two things: `SB_DISPLAY_ADDR`, an environment variable that determines the location of the shared memory; and `shmmxaddr`, a kernel configuration variable whose value determines the highest address at which shared memory can reside.

NOTE

To better understand Windows/9000 shared memory you need to read the “Shared Memory Usage” section of the “Resource Usage Considerations” appendix of the *HP Windows/9000 User’s Manual*.




Swap Space

Your HP-UX system has a fixed amount of secondary storage (swap space) reserved for swapped out processes. To run a large Lisp process you will need to increase the size of your swap space by approximately the difference between the size of the Lisp process you are running and the size of the Lisp process you wish to run. Swap space can be increased by adding a new disk or by reconfiguring one of your current disks to contain a smaller file system (and thus free some room for more swap space). It is easier to add a new disk than reconfigure an existing disk. These choices are discussed further in “Decisions” below.


NMODE Resource Usage

When you are determining what values to give kernel variables when you reconfigure, you need to consider the system resources used by the programs you will be running. The *HP-UX System Administrator Manual* has a worksheet to help you determine some of the kernel variable values, but there is some Lisp-specific information that is useful too.

If you’re running NMODE under Windows/9000, each NMODE window uses the following resources:

- 
- Two processes: the Windows/9000 *gserver* process, and a special NMODE *wdaemon* process.
 - Three *pty* master-slave pairs.
 - Two file descriptors.

If you use NMODE’s HP-UX access facility, each system shell or shell buffer uses the following resources:

- One shell process plus any processes (such as an executing command) that are spawned from that shell.
 - One *pty* master-slave pair.
 - Two file descriptors.
- 

Decisions

Before reconfiguring your system, you must make some decisions involving swap space, the maximum size of the process you want to run, and any other kernel changes you may want to make while you are reconfiguring.

Swap Configuration

Depending on how many disks your system has, there are several different possible configurations of swap space.

If you only have one disk, enlarging the size of your swap space will be more difficult. You will have to reinstall HP-UX and change the swap space size during installation. If you decide to do this, remember:

- Back up any files that are important to you before you reinstall
- The size of your file system will be reduced by the amount of increased swap space

If you have more than one disk, or will be adding a new disk, then you will have to decide which disk to use for the new swap space. Any particular disk can be used

- Solely as a file system
- Solely as swap space
- As a combination of file system and swap space

You should avoid changing your root disk since this requires reinstalling HP-UX. Most likely, your present configuration uses the root disk for file system and swap space, and a second disk for just file system. In this case, if you are not adding a new disk, you will have to reconfigure your second disk to change some file system space into swap space. You will have to back up any files on the disk before reconfiguring it. The *df(1M)* command will tell you how much free space is on the disk. Use this, and any knowledge about what files you may be adding or deleting, to decide how much of your file system space to convert to swap space.

An easier way to expand your swap space is by adding a disk. Then you will only have to decide if you want to use any of it for file system, and if so, how much.

Process Size

Before you reconfigure, you should know the size of the largest process you are planning to run. This determines (or is determined by) the amount of swap space necessary. It also determines the new values of some kernel variables you will be changing.

Kernel Configuration Variables

Four kernel configuration variables specify the sizes of various memory management data structures. These must be changed when you reconfigure. Table D.1 in Appendix D of the *HP-UX System Administrator Manual* shows these variables and their values based on ranges of maximum process size. We will be assuming a maximum process size of between forty and eighty megabytes, which means that the variables and their new values are

<code>dmmin</code>	64
<code>dmmax</code>	8192
<code>dmtext</code>	8192
<code>dmshm</code>	8192

Other kernel variables must also change, but their values are more flexible. They will be discussed along with the actual configuration process.

Maximum Process Size and Swap Space

The *HP-UX System Administrator Manual* gives a formula for determining how much swap space you will need based on several parameters. A simpler method is to simply add as much swap space as the size of the Lisp process you plan to run. In other words, if you want to run a forty megabyte Lisp process, add forty megabytes of swap space. With this method you may end up with a little extra swap space.

Since processes may increase the size of their data segments with calls to *malloc(3C)*, the amount of necessary swap space is not static. It is possible to have enough swap space one day and not enough the next because some background process allocated itself more space.

Other Configuration Changes

Before you reconfigure your system to allow running a big process, you may want to check if there are any other kernel configuration changes you'd like to make at the same time. There's no sense in reconfiguring again in a few weeks or months just because you weren't aware of a restriction of your current kernel. Appendix D of the *HP-UX System Administrator Manual* gives descriptions of all the kernel configuration variables. These include things like the total number of possible processes. Most of them are fairly esoteric, but you should skim through them to see if there are any that you want to change. You should also look at the previous section "NMODE Resource Usage" and the "Resource Usage Considerations" appendix of the *HP Windows/9000 User's Manual* to see if you may need to increase the kernel limit on a particular type of resource.

Procedure

The *HP-UX System Administrator Manual* contains a lengthy section on the use of *config* in the "Configuring HP-UX" section of the "Toolbox" chapter. This documentation assumes that you have read that section. You may not actually need to read it, but it wouldn't hurt.

Overview

In brief, the steps you will need to perform the reconfiguration are:

1. Make a file system on the disk that you will be using for additional swap space. If you have a disk that will be dedicated solely to swapping, you can skip this step.
2. Create a configuration file (*dfile*) that specifies the kernel configuration you desire.
3. Execute the *config(1M)* command on the *dfile* you created.
4. Execute the *make(1)* command with the makefile that *config* created. This will compile a new kernel for you.
5. Copy your old kernel (*/hp-ux*) to */SYSBCKUP*. This provides a means of recovering if your new kernel doesn't work.
6. Copy the kernel that you made in step 3 to */hp-ux*.
7. Reboot.
8. If you have swap space that is not on your root disk, activate it with the *swapon(1M)* command. To automate this step so that it happens any time you reboot, add a line to */etc/rc* that executes the command upon entry to state 1.
9. If you need to move the Windows/9000 shared memory, set the value of the *SB_DISPLAY_ADDR* environment variable appropriately. Appropriate values are discussed later.

Making a File System

If you have a disk that will be dedicated solely to swapping, you can skip this step. If you want to use a new disk for swapping and a file system, or you want to reconfigure a disk to provide more swap space, then you will have to build a new file system on the disk. Remember that making a new file system destroys any files previously stored on the disk, so make sure you have backups.

The *newfs(1M)* command uses information from the file `/etc/disktab` to create a new file system with the *mkfs(1M)* command. The file `/etc/disktab` has entries giving the values of arguments to *mkfs* for particular models of disks and amounts of swap space. There probably won't be an entry for the exact configuration you desire; you will have to add a new entry to `/etc/disktab` for your desired disk configuration.

To add a new entry, find one of the existing entries for the model of disk drive you are configuring. It will look something like this

```
hp7945:\
:ty=winchester:ns#8:nt#7:nc#786:\
:pa#44016:ba#8192:fa#1024:\
:se#1024:rm#3600:
```

This entry specifies an HP7945 disk drive with approximately 10 megabytes of swap space. Copy this, and then change the `nc` and `pa` fields. The number after `nc` is the number of cylinders to use in the disk's file system. The number after `pa` = `ns` × `nt` × `nc`.

There are 56K (`ns` times `nt` times `se`) bytes on each cylinder of a 7945, and a total of 968 cylinders. Let's say we want half the disk for swapping and half for file space. Our entry in `/etc/disktab` for this configuration would look like this

```
hp7945_27:\
:ty=winchester:ns#8:nt#7:nc#484:\
:pa#27104:ba#8192:fa#1024:\
:se#1024:rm#3600:
```

After you have created the entry in `/etc/disktab` with a unique name, run the *newfs* command

```
newfs -v /dev/rhd2 hp7945_27
```

The device file you specify in the command line should correspond to the character special file of the disk you are creating the file system on.

Creating a Configuration File

The *config(1M)* command uses information from a file (a *dfile*) to create a program and makefile for compiling a new kernel. The lines in the *dfile* can be various things, such as

- The names of device drivers to be included in the kernel
- Kernel configuration parameters and their values
- Specifications of swap devices

There are sample configuration files in the */etc/conf* directory. Copy one (*dfile.full.lan* is the most general) to another file and then edit the new file to make your configuration changes.

Kernel Configuration Variables

To change the value of a kernel parameter, add a line to the *dfile* with the name of the parameter followed by its new value.

The parameters you need to change to run a larger-than-normal process are: *dmmin*, *dmmax*, *dmttext*, *dmshm*, *maxdsiz*, *maxssiz*, and *shmmmaxaddr*. There may be other parameters you wish to change for other reasons.

The first four of these kernel variables are rigidly determined by the size of the largest process you wish to run. Table D.1 in appendix D of the *HP-UX System Administrator Manual* shows the appropriate values in terms of the process size. We'll assume here that the largest process you wish to run is between forty and eighty megabytes.

The values of the other variables are more flexible because they are used for error checking rather than resource allocation. The value of *maxdsiz* (maximum data segment size) should be at least as big as the size of the largest Lisp process you would like to run. The default value of *maxssiz* (maximum stack size) is adequate for many applications, but you may want to increase it for more flexibility. If your large Lisp process will be using Windows/9000, you need to set *shmmmaxaddr* high enough for the Windows/9000 shared memory to be above the data space of your Lisp process. A general guide for *shmmmaxaddr* is to set it to the size of the Lisp process you want to run plus four or five megabytes. Remember that *shmmmaxaddr* doesn't control where the shared memory actually is, only how high it can go in memory. You will have to set the *SB_DISPLAY_ADDR* environment variable to change the location of the window system's shared memory.

Specifying Swap Devices

You must specify in your *dfile* the swap device(s) that your system will use. Each device has a one-line entry of the form:

```
swap driver-name address swap-location [swap-size]
```

The fields are as follows:

<i>driver-name</i>	The swap device's driver name (e.g. <code>cs80</code>). The file <code>/etc/master</code> contains mappings of product number to device driver name.
<i>address</i>	The swap device's minor number in hexadecimal.
<i>swap-location</i>	Swap area's location in decimal. If it's <code>-1</code> , then the swap space is put on the disk after the file system. If it's <code>0</code> , then the whole disk is used for swapping. If it's greater than or equal to <code>1</code> , then a <i>disk-size</i> - <i>swap-location</i> sized swap area is reserved at the end of the disk.
<i>swap-size</i>	This optional field indicates the size of the swap area in 1K byte units.

A Sample Dfile

The following *dfile* specifies a configuration for a kernel with a full arsenal of device drivers that can run processes up to eighty megabytes. The swap space is on two disks: the root disk (after a file system), and a disk on bus address 2 that is dedicated entirely to swapping.

```
* This is the configuration file for a full system, with LAN
* drivers
cs80
flex
amigo
tape
printer
stape
srm
rje
ptymas
ptyslv
ieee802
ethernet
hpib
gpio
ciper
* cards
98624
98626
98626
```

```

98628
98642
* stuff for big processes (<= 80M)
maxdsiz 0x10000000
maxssiz 0x01000000
shmmxaddr 0x0ffffff
dmmin 64
dmmax 8192
dmtext 8192
dmsm 8192
* swap configuration
swap cs80 E0000 -1
swap cs80 E0200 0 54208

```

Running Config

Assume that you have created a *dfile* called *dfile.bigproc* in the */etc/conf* directory. The following commands would be executed to create and run a new kernel configuration. You must be super-user to perform these steps. This procedure is discussed in more detail in the *HP-UX System Administrator Manual*.

```

cd /etc/conf
config dfile.bigproc
make -f config.mk
cp /hp-ux /SYSBACKUP          # save the old kernel in case of trouble
cp hp-ux /hp-ux
exec reboot

```

After Booting

If you have swap space on more than one disc, this space must be enabled with *swapon(1M)* each time you reboot. For instance, for the example configuration given in the *dfile* above, you would execute the command

```
swapon /dev/hd2
```

You can automate this by adding the command to your */etc/rc* file so that it gets executed upon entry to state 2.

If your large process uses Windows/9000, you must change the value of the *SB_DISPLAY_ADDR* environment variable before starting the window manager. This must be large enough so that all of the shared memory is above the data space of your process. A general formula for computing a good value for *SB_DISPLAY_ADDR* is

```
SB_DISPLAY_ADDR = sizeofprocess + WMSHMSC + 0x100000
```


To change `SB_DISPLAY_ADDR` if you use the Bourne shell (*sh*), execute

```
SB_DISPLAY_ADDR=0x3800000  
export SB_DISPLAY_ADDR
```

To change `SB_DISPLAY_ADDR` if you use the C-shell (*csh*), execute

```
setenv SB_DISPLAY_ADDR 0x3800000
```







Chapter 3

The Example

Introduction

A common complaint about computer documentation is that it does not present useful examples. Many times this is because to be useful, the example would be too big. For this manual, there is one major example that illustrates many of the techniques that Lisp applications writers will need to use. This approach sacrifices modularity for comprehensiveness. It is hoped that seeing the program presented here as a model will make your programming task easier.



This chapter is a general introduction to the example program. Details of various aspects of the program are covered in the appropriate chapters. The source code for the program is given at the end of this chapter. You probably want to remove those pages and keep them close at hand so that they may easily be referred to when necessary. You could also print a listing yourself, since the code for the program was provided with your system in the `$LISP/doc/examples` directory.

The Program

The model application presented in this manual is an object-oriented tic-tac-toe game. The source code for this application is in the directory `$LISP/doc/examples`. The relevant files are

- `tictactoe.l` Lisp source for the game.
- `tictactoe.b` Binary for the game. You will have to create this yourself with `(compile-file "$LISP/doc/examples/tictactoe.l")`.
- `windexutil.l` Lisp source for utilities that are useful when using the Windows/9000 library.
- `windexutil.b` Binary for the window utilities. You will have to create this yourself with `(compile-file "$LISP/doc/examples/windowutil.l")`.

The directory `$LISP/doc/examples` contains a few other files discussed in this and other manuals.

Overview

The game is started by calling the function `tic-tac-toe` (in the `ttt` package). The program creates a graphics window in which it displays the tic-tac-toe board. The user is then prompted to answer a couple of questions by positioning the window locator over yes or no selection boxes and pressing a mouse button. The user moves by pressing any mouse button while the locator is in the square where they want to move. When the game is over, the user is asked whether or not they wish to play again. If they answer no, then the window is destroyed, and `tic-tac-toe` returns.

To load and run the game, execute the following forms.

```
(load "$LISP/doc/examples/tictactoe")
(ttt:tic-tac-toe)
```

Instance Types

There are two instance types used in the program. The `ttt-game` instance type defines the methods for interacting with the game window and for running the game. The `ttt-board` instance type maintains the state of the tic-tac-toe board and provides methods to access that state.

The moves of the two players are determined by the `ttt-game` instance variables `x-player-move` and `o-player-move`. The values of these instance variables are the names of `ttt-game` methods that determine what move to make and then make it. Only two such methods are provided: `:get-user-move` (which prompts the user for his move and gets it) and `:make-move` (the routine that moves for the “computer”). It could be argued that the methods for making moves should really be `ttt-board` methods, but since `:get-user-move` must interact with the game window, they were made `ttt-game` methods. This works as long as any particular implementation of `ttt-board` provides a standard set of methods for obtaining information about the state of the game.

The TTT-BOARD Instance Type

The state of the board is represented in an instance of `ttt-board`. There is a vector (`board-array`) with nine slots for the nine squares on the board. The squares are numbered as shown in this diagram.

0		1		2

3		4		5

6		7		8

Note that while the board is represented as a one-dimensional array, the `ttt-board` methods provide an interface that makes it look like a matrix (ie. `i` and `j` coordinates).

The other data structure in the `ttt-board` instance type is `free-spaces-list`, a list of all the squares on the board which are not yet occupied. This provides an easy way of checking for forced moves in the methods `:winning-move` and `:defending-move`.

You are encouraged to experiment with the definition of `ttt-board` to provide a different representation of the board and a richer set of methods for obtaining information. This can serve as a platform for a “smarter” method for making the “computer” move. You might also generalize it so that it can represent an `n` by `n` board (it’s easier if you restrict `n` to be odd).

Windowing Utilities

The tic-tac-toe program uses the Windows/9000 libraries. Some Lisp functions that are useful for using Windows/9000 are defined in the windowutil module in the \$LISP/doc/examples directory. These routines correspond to the C routines for establishing and terminating window communication presented in the *HP Windows/9000 Programmer's Manual*.

Source Code

The source code for the tic-tac-toe program follows.

```
.....  
;  
; File:          tictactoe.l  
; SCCS:         @(#) $hi/doc/examples/tictactoe.l 1.1@(#) 4/23/86 13:56:34  
; Description:  Tic Tac Toe Game  
; Language:    Lisp  
; Package:     TTT  
;  
; (c) Copyright 1986, Hewlett-Packard Company, all rights reserved.  
;  
.....  
  
(provide "ttt")  
  
(in-package 'ttt)  
  
;;; Shadowing is done in the shadowing-import below  
  
(export '(tic-tac-toe))  
  
(require "objects")  
(require "hp-ux_3g")  
(require "windowutil" "$LISP/doc/examples/windowutil")  
(require "exception")  
  
(shadowing-import '(hp-ux_3g:PUSH hp-ux_3g:REPLACE))  
(use-package '(lisp extn hp-ux_3g hp-ux_3w windowutil))
```

```
.....  
;;;  
;;; Definition and methods for the tictactoe game object  
;;;  
.....
```

```
;;;  
;;; The ttt-game instance type maintains all the info needed to communicate  
;;; with the game window. Its methods run the game.  
;;;
```

```
(define-type ttt-game  
  (:var board) ; An instance of ttt-board  
  (:var window-path) ; The HP-UX pathname of the window special  
device  
  (:var wm-fildes ; File descriptor of window manager interface  
    (:init nil))  
  (:var gr-fildes ; File descriptor of graphics window  
    (:init nil))  
  (:var marker-font-id  
    (:init nil))  
  (:var prompt-font-id  
    (:init nil))  
  (:var x-player-move) ; Method to generate x's move  
  (:var o-player-move) ; Method to generate o's move  
  (:var created-okay ; Flag for valid creation of window  
    (:init nil) :gettable)  
  :all-initable)
```

```

;;;
;;; Initialize the game. This method does all the work of
;;; setting up the window interface.
;;;
(define-method (ttt-game :init) (keylist)
  (extn:when-error
    (and
      (setq wm-fildes (establish-wm-communication))
      (setf window-path (make-string 25))
      (hp-ux-return (wmpathmake "WMDIR" "ttt" window-path))
      ;; Create Window
      (hp-ux-return (wcreate_graphics wm-fildes
                                     window-path
                                     700
                                     100
                                     180
                                     225
                                     180
                                     225
                                     SETRETAIN
                                     SETNOBANNER))
      (setq gr-fildes (establish-gr-communication wm-fildes window-path))
      (setq marker-font-id
        (let (font-id)
          (declare (special font-id))
          (if (hp-ux-return (fm_load gr-fildes "/usr/lib/raster/18x30/pica.8U"
'font-id))
              font-id)))
      (setq prompt-font-id
        (let (font-id)
          (declare (special font-id))
          (if (hp-ux-return (fm_load gr-fildes "/usr/lib/raster/7x10/lp.8U"
'font-id))
              font-id)))
      (setf created-okay t)
      )
    ;; Error handling forms
    (format t "Error in creation function ~A; called with ~A~%"
      (extn:exception-signaller)
      (extn:exception-arguments))
    (format t "Cleaning up...~%"
      (=> self :cleanup)
    )
  )
)

```



```
;;;
;;; :Move-x-player just calls the method to make x's move and updates the
display.
```

```
;;;
(define-method (ttt-game :move-x-player) ()
  (=> self x-player-move #\X)
  (=> self :draw-markers)
)
```

```
;;;
;;; :Move-o-player just calls the method to make o's move and updates the
display.
```

```
;;;
(define-method (ttt-game :move-o-player) ()
  (=> self o-player-move #\O)
  (=> self :draw-markers)
)
```

```
;;;
;;; :GameOverp returns true when the current game is over.
```

```
;;;
(define-method (ttt-game :gameoverp) ()
  (or (=> board :fullp)
      (=> board :winnerp)
  )
)
```

```
;;;
;;; :Play-again-p returns true if the user indicates that he wants to play again.
```

```
;;;
(define-method (ttt-game :play-again-p) ()
  (=> self :prompt-user "Like to try again?")
  (=> self :user-y-or-n-p)
)
```

```

:-----:
:
:;;
:;; Methods for dealing with the game display.
:;;
:-----:

```

```

:;;
:;; :Prompt-user clears the prompt area and then writes a string to it.
:;; The string should be less than 38 characters long.
:;;

```

```

(define-method (ttt-game :prompt-user) (prompt)
  (=> self :clear-prompt-area)
  (fm_activate gr-fildes prompt-font-id)
  (fm_write gr-fildes 0 187 prompt (length prompt) TRUE TRUE)
)

```

```

:;;
:;; :Clear-prompt-area clears the prompt area.
:;;

```

```

(define-method (ttt-game :clear-prompt-area) ()
  (let ((blankstring "
                                "))
    (fm_activate gr-fildes prompt-font-id)
    (fm_write gr-fildes 0 187 blankstring 38 TRUE TRUE)
  )
)

```

```

:;;
:;; :User-y-or-n-p displays yes and no boxes in the prompt area, returns
:;; true if the user boinks (any button) with the locator in the yes box,
:;; nil otherwise.
:;;

```

```

(define-method (ttt-game :user-y-or-n-p) ()
  (wsetlocator gr-fildes -100 100)
  (second (make-and-activate-menu gr-fildes "Yes or No"
                                   '(("Yes" yes) ("No" nil))))
)

```

```
;;;
;;; :Draw-grid draws the crossed lines for the game display.
;;;
```

```
(define-method (ttt-game :draw-grid) ()
  (dcmove gr-fildes 0 59)
  (dcdraw gr-fildes 179 59)
  (dcmove gr-fildes 179 119)
  (dcdraw gr-fildes 0 119)
  (dcmove gr-fildes 0 179)
  (dcdraw gr-fildes 179 179)
  (dcmove gr-fildes 119 179)
  (dcdraw gr-fildes 119 0)
  (dcmove gr-fildes 59 0)
  (dcdraw gr-fildes 59 179)
  (make_picture_current gr-fildes)
)
```

```
;;;
;;; :Draw-markers uses fm_write to draw the markers on the board display.
;;;
```

```
(define-method (ttt-game :draw-markers) ()
  (fm_activate gr-fildes marker-font-id)
  (dotimes (i 3)
    (dotimes (j 3)
      (fm_write gr-fildes (+ (* i 60) 20) (+ (* j 60) 14)
        (string (=> board :get-square i j)) 1 TRUE TRUE)
    )
  )
)
```

```
;;;
;;; :Cleanup "undoes" all the game's connections to Windows/9000
;;;
(define-method (ttt-game :cleanup) ()
  (when marker-font-id
    (fm_activate gr-fildes marker-font-id)
    (fm_remove gr-fildes marker-font-id))
  (when marker-prompt-id
    (fm_activate gr-fildes prompt-font-id)
    (fm_remove gr-fildes prompt-font-id))
  (when gr-fildes
    (terminate-gr-communication gr-fildes))
  (when wm-fildes
    (wdestroy wm-fildes window-path)
    (terminate-wm-communication wm-fildes))
  )
```

```

:.....:
:....:
:;;; Methods for making moves (names are suitable
:;;; values for x-player-move and o-player-move.
:....:
:.....:

:;;
:;;; :Get-user-move is one of the valid methods for x-player-move and
:;;; o-player-move. It prompts the user, then waits for them to boink
:;;; on the square where they wish to move.
:;;
(define-method (ttt-game :get-user-move) (marker)
  (=> self :prompt-user (format nil "Player ~A Move" marker))
  (prog ()
    start
    (let* ((points (get-boink gr-fildes))
           (x (first points))
           (y (second points))
           (i (truncate x 60))
           (j (truncate y 60))
          )
      (=> self :clear-prompt-area)
      (if (or (not (and (<= 0 i 2) (<= 0 j 2))) (= board :occupiedp i j))
          (progn (= self :prompt-user "Invalid: Try again")
                 (go start))
          ;; Else
          (=> board :set-square i j marker)
          ))
    )
  )
)

```

```

;;;
;;; :Make-move is a brain-damaged method that makes moves for the "computer"
;;;
(define-method (ttt-game :make-move) (marker)
  (let (move)
    (cond ((setf move (=> board :winning-move marker))
          (=> board :set-square (first move) (second move) marker))
          ((setf move (=> board :defending-move marker))
          (=> board :set-square (first move) (second move) marker))
          (t (setf move (=> board :default-move))
            (=> board :set-square (first move) (second move) marker)))
    )
  )
)

```

```

:-----:
:-----:
:-----:
:-----:
:-----:
:-----:
:-----:

```

```

;;;
;;; The ttt-board instance type maintains the state of the game board
;;; and provides methods to update and access that state.
;;;
(define-type ttt-board
  (:var board-array)
  (:var free-spaces-list)
)

```

```

;;;
;;; Initializes the two data structures.
;;;
(define-method (ttt-board :init) (keylist)
  (setq board-array (make-array 9 :initial-element nil))
  (setq free-spaces-list (list 4 0 6 2 8 3 1 7 5))
)

```

```

;;;
;;; Set a given square on the board to a given marker.
;;;
(define-method (ttt-board :set-square) (i j marker)
  (let ((index (+ i (* 3 j))))
    (setf (svref board-array index) marker)
    (setf free-spaces-list (remove index free-spaces-list)))
  )
)

```

```

;;;
;;; Return the character for the marker at a given position on the board.
;;; Note that this returns a space character for unoccupied squares.
;;;
(define-method (ttt-board :get-square) (i j)
  (let ((square (svref board-array (+ i (* 3 j)))))
    (cond (square)
          (t #\Space)))
  )
)

```

```

;;;
;;; Returns true if a given square is occupied, otherwise nil.
;;;
(define-method (ttt-board :occupiedp) (i j)
  (svref board-array (+ i (* 3 j)))
)
)

```

```

;;;
;;; Returns true if there are no unoccupied squares, otherwise nil.
;;;
(define-method (ttt-board :fullp) ()
  (null free-spaces-list)
)
)

```

```

;;;
;;; Laboriously churns through the various ways of getting three in a row.
;;; Returns the marker of the winner if there is one, otherwise nil.
;;;
(define-method (ttt-board :winnerp) ()
  (catch 'done
    ;; Check 0 1 2 and 0 3 6
    (let ((upper-left (svref board-array 0))
          (middle (svref board-array 4))
          (lower-right (svref board-array 8)))
      (if (and upper-left
                (or (and (equalp upper-left (svref board-array 1))
                          (equalp upper-left (svref board-array 2)))
                    (and (equalp upper-left (svref board-array 3))
                          (equalp upper-left (svref board-array 6)))
                ))
          (throw 'done upper-left))
        ;; Check 1 4 7, 3 4 5, 0 4 8, and 2 4 6
        (if (and middle
                  (or (and (equalp middle (svref board-array 1))
                            (equalp middle (svref board-array 7)))
                      (and (equalp middle (svref board-array 3))
                            (equalp middle (svref board-array 5)))
                      (and (equalp middle upper-left)
                            (equalp middle lower-right))
                      (and (equalp middle (svref board-array 2))
                            (equalp middle (svref board-array 6)))
                  ))
            (throw 'done middle))
          ;; Check 2 5 8 and 6 7 8
          (if (and lower-right
                    (or (and (equalp lower-right (svref board-array 2))
                              (equalp lower-right (svref board-array 5)))
                        (and (equalp lower-right (svref board-array 6))
                              (equalp lower-right (svref board-array 7)))
                    ))
              (throw 'done lower-right)))
            ))
  ))

```



```

;;;
;;; Tries all available moves and returns a winning move for marker
;;; if there is one. Otherwise it returns nil.
;;;
(define-method (ttt-board :winning-move) (marker)
  (catch 'done
    (dolist (potential-move free-spaces-list)
      (unwind-protect
        (progn
          (setf (svref board-array potential-move) marker)
          (if (=> self :winnerp)
              (throw 'done (nreverse
                           (multiple-value-list (floor potential-move 3))))
              ))
          ;; Protect Form
          (setf (svref board-array potential-move) nil)
        )
      )
    )
  )
)

```

```

;;;
;;; Tries available moves to see if opponent of marker can win with next move.
;;; Returns that move if there is one, otherwise nil.
;;;
(define-method (ttt-board :defending-move) (marker)
  (let ((opp-marker (case marker
                     (#\X #\O)
                     (#\O #\X))))
    (catch 'done
      (dolist (potential-move free-spaces-list)
        (unwind-protect
          (progn
            (setf (svref board-array potential-move) opp-marker)
            (if (=> self :winnerp)
                (throw 'done (nreverse
                             (multiple-value-list (floor potential-move 3))))
                ))
            ;; Protect Form
            (setf (svref board-array potential-move) nil)
          )
        )
      )
    )
  )
)

```

```
;;;
;;; Returns first move of free list
;;;
(define-method (ttt-board :default-move) ()
  (nreverse (multiple-value-list (floor (first free-spaces-list) 3)))
  )
```

```

.....
;;;
;;; Functions
;;;
.....

```

```

;;;
;;; The main loop that runs the game
;;;
(defun tic-tac-toe ()
  (let ((*game* (make-instance 'ttt-game)))
    (declare (special *game*))
    (when (=> *game* :created-okay)
      (catch 'end-of-game
        (unwind-protect
          (loop
            (catch 'another
              (=> *game* :start)
              (loop
                (=> *game* :move-x-player)
                (end-of-game? *game*)
                (=> *game* :move-o-player)
                (end-of-game? *game*)
              ))
            )
          )
        (=> *game* :cleanup))
      )
    )
  )
)

```

```

;;;
;;; Checks to see if the game is done. If it is, asks
;;; user if they want to play again and reacts accordingly.
;;;
(defun end-of-game? (game)
  (if (=> game :gameoverp)
    (if (=> game :play-again-p)
      (throw 'another nil)
      (throw 'end-of-game 'Done)
    )
  )
)
)

```

Chapter 4

Libraries

Introduction

The non-Lisp function calling mechanism makes it possible for you to call HP-UX library or system functions from Lisp. It would however, be less than friendly if you had to define all the access functions yourself. For this reason (and because we wanted to use them too), HP has provided Lisp functions that call many of the HP-UX library routines.

This chapter lists all the available functions for calling the library routines and the types of arguments that they require, but does not explain what the functions do. See the appropriate HP-UX documentation for that information. The object code files for the functions described here are in the directory `$LISP/modules/lib`.

Pointer Parameters

Some of the library functions expect arguments that are pointers. The non-Lisp function calling mechanism requires that the corresponding argument to the Lisp access function be a symbol. The global value cell of the symbol is used to transmit the value. See “Calling Non-Lisp Routines” in the *Lisp Programmer’s Guide*.

Constants

When calling HP-UX library routines from C, you normally use the include facility to insert the text of various “include” files into the source of your program. These files contain definitions for constants that are convenient to use as arguments to certain library functions. Many of these constants have been defined for you in the modules that provide the Lisp access functions. The names of predefined constants are listed in the section that describes the functions they are most often used with.

HP-UX System Calls

There are predefined Lisp access functions for all of the HP-UX system functions described in Section 2 of the *HP-UX Reference*. These are in the `hp-ux_2` module, which can be conditionally loaded with

```
(require "hp-ux_2")
```

All of the functions listed here are in the `hp-ux_2` package. Because of name conflicts with some Common Lisp functions, we recommend that anyone using these functions import the desired symbols or qualify the names (with `hp-ux_2:`), instead of using the package directly.

Special Considerations

Some of the functions in this section have special considerations because their corresponding C functions expect arguments whose types have no convenient equivalent in Lisp. This has been handled a couple different ways.

Structure Arguments

For some of the section 2 functions that take structure arguments, we have defined equivalent Lisp structures to be used as arguments to the Lisp version of the function. These functions and structure definitions are described below. Remember that they are defined in the `hp-ux_2` package.

For the functions `gettimeofday` and `settimeofday`, the structure definitions are:

```
(defstruct (timeval (:type (vector integer)))  
  (tv_sec 0)  
  (tv_usec 0))
```

```
(defstruct (timezone (:type (vector integer)))  
  (tz_minuteswest 0)  
  (tz_dsttime 0))
```

For the `times` function, the structure definition is:

```
(defstruct (tms  
  (:type (vector integer)))  
  (utime 0)  
  (stime 0)  
  (cutime 0)  
  (cstime 0))
```

For the `uname` function, the structure definition is:

```
(defstruct utsname
  (sysname "")
  (nodename "")
  (release "")
  (version "")
  (machine "")
  (idnumber ""))
```

For the `utime` function, the structure definition is:

```
(defstruct (utimbuf
  (:type (vector integer)))
  (actime 0)
  (modtime 0))
```

If you want to use the `ioctl` function to change the `termio` structure of a terminal device, we have supplied the Lisp function `tty-ioctl`

`(hp-ux_2:tty-ioctl file-descriptor command arg)` *Function*

If *command* is one of `TCGETA`, `TCSETA`, `TCSETAW`, or `TCSETAF`, then *arg* should be a `termio` structure. The definition of the structure is

```
(defstruct termio
  (c_iflag 0 :type integer)
  (c_oflag 0 :type integer)
  (c_cflag 0 :type integer)
  (c_lflag 0 :type integer)
  (c_line #\a :type string-char)
  (c_cc " " :type (array string-char)))
```

Addresses

Some section 2 functions take arguments that are supposed to be addresses. At the moment, there is no support in Lisp for easily generating a meaningful address for these functions. If you use such a function, you are responsible for making sure that you pass it a meaningful argument.

For some of these functions, it makes more sense to write a C program that calls the system function, and then call your C function from Lisp using the non-Lisp function calling facility.

All of the non-trivial function calls are marked ; `non-trivial` in the list below.

Constants

TCGETA
TCSETA
TCSETAW
TCSETAF
TCSBRK
TCXONC
TCFLSH

Functions

(access *character-array integer*)

(acct *character-array*)

(alarm *integer*)

(brk *integer*)

(sbrk *integer*)

(chdir *character-array*)

(chmod *character-array integer*)

(chown *character-array integer integer*)

(chroot *character-array*)

(close *integer*)

(creat *character-array integer*)

(dup *integer*)

(errno)

For `execl`, do not specify the last argument as 0; it is done for you.

(`execl` *character-array &rest character-arrays*)

(`execv` *character-array vector-of-strings*)

For `execl`, do not specify the second to last argument as 0; it is done for you.

(`execl` *character-array &rest character-arrays vector-of-strings*)

(`execve` *character-array vector-of-strings vector-of-strings*)

For `execlp`, do not specify the last argument as 0; it is done for you.

(`execlp` *character-array &rest character-arrays*)

(execvp *character-array vector-of-strings*)

(exit *integer*)

(_exit *integer*)

(fcntl *integer integer integer*)

(fork)

(fsync *integer*)

(ftime *vector*) ; non-trivial

(gethostname *character-array integer*)

(getitimer *integer vector*) ; non-trivial

(setitimer *integer vector vector*) ; non-trivial

(getpid)

(getpgrp)

(getppid)

(getprivgrp *array*)

(setprivgrp *integer array*)

(gettimeofday *timeval-structure timezone-structure*)

(settimeofday *timeval-structure timezone-structure*)

(getuid)

(geteuid)

(getgid)

(getegid)

(ioctl *integer integer fixnum*) ; See tty-ioctl, described above.

(kill *integer integer*)

(link *character-array character-array*)

(lockf *integer integer integer*)

(lseek *integer integer integer*)
(mkdir *character-array integer*)
(mknod *character-array integer integer*)
(mount *character-array character-array integer*)
(msgctl *integer integer array*) ; non-trivial
(msgget *integer integer*)
(msgsnd *integer array integer integer*) ; non-trivial
(msgrcv *integer array integer integer integer*) ; non-trivial
(nice *integer*)
(open *character-array integer &optional integer*)
(pause)
(pipe *two-element-integer-vector*)
(plock *integer*)
(prealloc *integer integer*)
(profil *integer integer integer integer*)
(ptrace *integer integer integer integer*)
(read *integer character-array integer*)
(readv *integer array integer*) ; non-trivial
(reboot *integer character-array character-array*)
(rmdir *character-array*)
(rtprio *integer integer*)
(select *integer integer-array integer-array integer-array array*) ; non-trivial
(semctl *integer integer integer array*) ; non-trivial
(semget *integer integer integer*)
(semop *integer array integer*) ; non-trivial

(setgroups *integer integer-array*)
(sethostname *character-array integer*)
(setpgrp)
(setuid *integer*)
(setgid *integer*)
(setprgrp)
(setuid *integer*)
(shmctl *integer integer array*) ; non-trivial
(shmget *integer integer integer*)
(shmat *integer integer integer*) ; non-trivial
(shmdt *integer*) ; non-trivial
(signal *integer integer*) ; non-trivial
(sigpause *integer*) ; non-trivial
(sigsetmask *integer*) ; non-trivial
(sigspace *integer*) ; non-trivial
(sigvector *integer array array*) ; non-trivial
(stat *character-array array*) ; non-trivial
(fstat *integer array*) ; non-trivial
(stime *integer*)
(stty *integer array*) ; non-trivial
(gtty *integer array*) ; non-trivial
(sync)
(time *symbol*) ; Value cell of the symbol will be used
(times *tms-structure*)
(truncate *character-array integer*)

(ftruncate *integer integer*)
(ulimit *integer integer*)
(umask *integer*)
(umount *character-array*)
(uname *utsname-structure*)
(unlink *character-array*)
(ustat *integer array*) ; non-trivial
(utime *character-array utimbuf-structure*)
(vfork)
(wait *symbol*) ; The value cell of the symbol will be used.
(write *integer character-array integer*)

Windows/9000 Functions

The Lisp access functions for Windows/9000 are defined in the `hp-ux_3w` module, which can be conditionally loaded with

```
(require "hp-ux_3w")
```

All of the symbols described here are interned in the `hp-ux_3w` package.

Constants

The following useful constants are defined in the `hp-ux_3w` module.

COLORMODE	FAGREEN	MENU_SELECTABLE
DFLTNAMEMAX	FAHALFBRIGHT	MENU_SEPARATOR
DFLT_IPOS	FAINVERSE	MENU_STRING
DFLT_NAME	FAOFF	MENU_TRACKINV
DFLT_WPOS	FAPLANE	MENU_TRACKNOCHNG
DO_MC	FARED	NOMCONCFLAGFALSE
ECHO_ALPHA	FAROLLDOWN	NOMCONCFLAGTRUE
ECHO_BEST	FAROLLLEFT	NOMCONCLEAR
ECHO_BOX	FAROLLRIGHT	NOMCONFARECTWRITE
ECHO_DEFAULT	FAROLLUP	NOMCONFAROLL
ECHO_DEVDEP	FASERIAL	NOMCONFWRITE
ECHO_FULL	FAUNDERLINE	SETAUTODESTROY
ECHO_NOOPT	FAWINDOW	SETAUTOSELECT
ECHO_OPT	FAWONB	SETAUTOTOP
ECHO_RUBLINE	FONTIDMAX	SETBANNER
ECHO_RUBRECT	FONTIDMIN	SETBOTTOM
ECHO_SMALL	GETAUTODESTROY	SETCONCEAL
ECHO_USERDEF	GETAUTOSELECT	SETICONIC
EVENT_B1_DOWN	GETAUTOTOP	SETNOAUTODESTROY
EVENT_B1_UP	GETBANNER	SETNOAUTOSELECT
EVENT_B2_DOWN	GETBOTTOM	SETNOAUTOTOP
EVENT_B2_UP	GETCONCEAL	SETNOBANNER
EVENT_B3_DOWN	GETFONTID	SETNOBOTTOM
EVENT_B3_UP	GETICONIC	SETNOCONCEAL
EVENT_B4_DOWN	GETPAUSE	SETNOICONIC
EVENT_B4_UP	GETRECOVER	SETNOPAUSE
EVENT_B5_DOWN	GETSELECT	SETNORECOVER
EVENT_B5_UP	GETTOP	SETNORETAIN
EVENT_B6_DOWN	IMODE_FILE	SETNOSELECT
EVENT_B6_UP	IMODE_NONE	SETNOTOP
EVENT_B7_DOWN	IMODE_TYPE	SETPAUSE
EVENT_B7_UP	LABELMAX	SETRECOVER
EVENT_B8_DOWN	LMODE_DISP	SETRETAIN
EVENT_B8_UP	LMODE_NONE	SETSELECT
EVENT_ECHO	MCALWAYS	SETTOP
EVENT_MENU	MENU_ACT_AUTO	SFKLABELMAX
EVENT_MOVE	MENU_ACT_DIS	SFKOFF
EVENT_REPAINT	MENU_ACT_IM	SFKON
EVENT_SELECT	MENU_ACT_INQ	SFKSEPOFF
EVENT_SIZE	MENU_DISP GREY	SFKSEPON
FABLINK	MENU_DISP NORM	SHUFFLEDOWN
FABLUE	MENU_NEWITEM	SHUFFLEUP
FABONW	MENU_NOPARENT	WINMAX
FACOLOR	MENU_NOTSELECTABLE	WINNAMEMAX
FACURSORNOMOVE	MENU_POPUP	WINUNITMAX

Functions

Below is a list of all the Lisp functions for accessing the Windows/9000 library functions. It gives the name of the functions and the types of arguments they require. See the Windows/9000 documentation for information on what each function does. When an argument is a symbol, the value cell of the symbol is used. For more information see the chapter "Calling Non-Lisp Routines" in the *Lisp Programmer's Guide*.

(*altfont_term0 fixnum fixnum*)

(*basefont_term0 fixnum fixnum*)

(*faclear fixnum fixnum four-element-fixnum-vector*)

(*facolors fixnum fixnum fixnum*)

(*facursor fixnum fixnum fixnum fixnum*)

(*fafontactivate fixnum fixnum*)

(*fafontload fixnum simple-string*)

(*fafontremove fixnum fixnum*)

(*fagetinfo fixnum fifteen-element-fixnum-vector*)

(*fainit fixnum fixnum*)

(*farectwrite fixnum fixnum fixnum four-element-fixnum-vector*)

(*faroll fixnum fixnum fixnum four-element-fixnum-vector*)

(*fasetinfo fixnum fifteen-element-fixnum-vector*)

(*faterminate fixnum*)

(*fawrite fixnum fixnum fixnum simple-string simple-string fixnum*)

(*fm_activate fixnum fixnum*)

(*fm_clipflag fixnum fixnum*)

(*fm_cliplim fixnum fixnum fixnum fixnum fixnum*)

(*fm_colors fixnum fixnum fixnum*)

(*fm_fileinfo simple-string*
 symbol ; The value cells of these
 symbol ; symbols will be used to
 symbol) ; return the information.

(fm_fontdir *fixnum fixnum*)

(fm_getname *fixnum fixnum simple-string*)

(fm_load *fixnum simple-string symbol*) ; Value cell of symbol will be used

(fm_opt *fixnum fixnum*)

(fm_rasterinfo *fixnum*
fixnum
symbol ; The value cells of these
symbol ; symbols will be used to
symbol) ; return the information.

(fm_remove *fixnum fixnum*)

(fm_str_len *fixnum simple-string fixnum*)

The Lisp interface to fm_styleinfo is not currently implemented.

(fm_write *fixnum*
fixnum
fixnum
simple-string
fixnum
fixnum
fixnum)

(fontgetid_term0 *fixnum simple-string*)

(fontgetname_term0 *fixnum fixnum*)

(fontload_term0 *fixnum simple-string*)

(fontreplaceall_term0 *fixnum simple-string simple-string*)

(fontsize_term0 *fixnum symbol symbol*) ; The value cells of the symbols are used.

(fontswap_term0 *fixnum simple-string fixnum*)

(fromxy_term0 *fixnum*
fixnum
fixnum
symbol ; The value cells of these
symbol) ; symbols will be used.

(toxy_term0 *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used.

fixnum
fixnum)

(*wautodestroy fixnum fixnum*)

(*wautoselect fixnum fixnum*)

(*wautotop fixnum fixnum*)

(*wbanner fixnum fixnum*)

(*wbottom fixnum fixnum*)

(*wconceal fixnum fixnum*)

(*wcreate_graphics fixnum*
simple-string
fixnum
fixnum
fixnum
fixnum
fixnum
fixnum
fixnum)

(*wcreate_term0 fixnum*
simple-string
fixnum
fixnum
fixnum
fixnum
fixnum
fixnum
fixnum
fixnum
simple-string
simple-string
fixnum
fixnum)

(*wdestroy fixnum simple-string*)

(*wdf1tpos fixnum*
fixnum
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.
symbol
simple-string)

(weventclear *fixnum* *fixnum*)

(weventpoll *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.
symbol)

(wgetbcolor *fixnum*
symbol ; The value cells of these symbols
symbol) ; will contain the returned values.

(wgetbcoords *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.
symbol)

(wgetcoords *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.
symbol
symbol
symbol
symbol
symbol)

(wgetecho *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.
symbol)

(wgeticonpos *fixnum*
symbol ; The value cells of these symbols
symbol) ; will contain the returned values.

(wgetlocator *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol) ; return values.

(wgetname *fixnum* *simple-string*)

(wgetrasterecho *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.

symbol
symbol
symbol
simple-string
simple-string)

(wgetscreen *fixnum*
symbol ; The value cells of these
symbol ; symbols will be used to
symbol ; return values.
symbol
symbol)

(wgetsigmask *fixnum symbol*) : The value cell of the symbol will be used.

(wiconic *fixnum fixnum*)

(winit *fixnum*)

(wmenu_create *fixnum fixnum fixnum fixnum fixnum*)

(wmenu_item *fixnum fixnum fixnum fixnum fixnum simple-string*)

(wmenu_delete *fixnum fixnum*)

(wmenu_activate *fixnum fixnum fixnum*)

(wmenu_eventread *fixnum symbol symbol*) ; The value cells of the symbols are used

(wminquire *fixnum simple-string simple-string*)

(wmkill *fixnum*)

(wmove *fixnum fixnum fixnum*)

(wpathmake *simple-string simple-string simple-string*)

(wmrepaint *fixnum*)

(wpan *fixnum fixnum fixnum*)

(wpauseoutput *fixnum fixnum*)

(wselect *fixnum fixnum*)

(wsetbcolor *fixnum fixnum fixnum*)

(wsetecho *fixnum fixnum fixnum fixnum fixnum*)

(wseticon *fixnum fixnum fixnum simple-string*)



(wseticonpos *fixnum fixnum fixnum*)

(wsetlabel *fixnum simple-string*)

(wsetlocator *fixnum fixnum fixnum*)

(wsetrasterecho *fixnum*
fixnum
fixnum
fixnum
fixnum
fixnum
simple-string
simple-string)

(wsetsigmask *fixnum fixnum*)

(wsfk_mode *fixnum fixnum*)

(wsfk_prog *fixnum fixnum simple-string string-character*)



(wshuffle *fixnum fixnum*)

(wsize *fixnum fixnum fixnum*)

(wterminate *fixnum*)

(wtop *fixnum fixnum*)



Starbase Graphics Functions

The Lisp access functions for Starbase graphics are defined in the `hp-ux_3g` module, which can be conditionally loaded with

```
(require "hp-ux_3g")
```

All of the symbols described here are interned in the `hp-ux_3g` package. There are two symbols in this package that conflict with symbols in the `lisp` package (`push` and `replace`). If you want to use both of these packages, you will need to precede the `use-package` with a call to `shadowing-import`. For instance,

```
(in-package 'user)
(shadowing-import '(hp-ux_3g:push hp-ux_3g:replace))
(use-package '(hp-ux_3g lisp))
```

After doing this, to call the Common Lisp functions `push` and `replace`, you will need to qualify the symbols with `lisp:` (e.g. `lisp:push`). Similarly, you could do the `shadowing-import` with `lisp:push` and `lisp:replace`, and then explicitly refer to `hp-ux_3g:push` and `hp-ux_3g:replace`.

Constants

The following useful constants are defined in the `hp-ux_3g` module.

ALL	HPTERM_640X400	R_BIT_MASK
BLINK_PLANES	HPTERM_PRINT_ESC	R_BIT_MODE
CENTER_DASH	HP_8BIT	R_DEF_FILL_PAT
CENTER_DASH_DASH	IGNORE_RELEASE	R_FREE_OFFSCREEN
CHARACTER_TEXT	INDEV	R_FULL_FRAME_BUFFER
CHOICE	INIT	R_GET_FRAME_BUFFER
CLEAR_CLIP_RECTANGLE	INMETA	R_GET_WINDOW_INFO
CLEAR_DISPLAY_SURFACE	INT_HOLLOW	R_LOCK_DEVICE
CLEAR_VDC_EXTENT	INT_SOLID	R_UNLOCK_DEVICE
CLIP_OFF	ISOTROPIC	
SIMULTANEOUS_EVENT_FOLLOWS		
CLIP_TO_RECT	ISO_7BIT	SINGLE_EVENT
CLIP_TO_VDC	ISO_8BIT	SOLID
DASH	LOCATOR	SPOOLED
DASH_DOT	LONG_DASH	STRING_TEXT
DASH_DOT_DOT	METRIC	STROKE_TEXT
DISABLE_AUTO_PROMPT	NOT_EMPTY_NO_OVERFLOW	SWITCH_SEMAPHORE
DISTORT	NOT_EMPTY_OVERFLOW	TA_BASE
DOT	NO_ERROR_PRINTING	TA_BOTTOM
EMPTY_NO_OVERFLOW	OUTDEV	TA_CAP
EMPTY_OVERFLOW	OUTINDEV	TA_CENTER
ENABLE_AUTO_PROMPT	OUTMETA	TA_CONTINUOUS_HORIZONTAL
FALSE	PATH_DOWN	TA_CONTINUOUS_VERTICAL
FRACTIONAL	PATH_LEFT	TA_HALF

GA_NONE	PATH_RIGHT	TA_LEFT
GB_NONE	PATH_UP	TA_NORMAL_HORIZONTAL
GKSM_GET_ITEM_TYPE	POST	TA_NORMAL_VERTICAL
GKSM_INQ_COLOR_NDCES	PRE	TA_RIGHT
GKSM_INQ_PAT_REP	PRINT_ERRORS	TA_TOP
GKSM_READ_ITEM	PRINT_WARNINGS	TC_HALF_PIXEL
GKSM_SKIP_ITEM	PROMPT_OFF	THREE_D
GKSM_WRITE_ITEM	PROMPT_ON	TOS_TEXT
HP26_PRINT_ESC	PUSH	TRIGGER_ON_RELEASE
HPGL_SET_PEN_NUM	READ_COLOR_MAP	TRUE
HPGL_SET_PEN_SPEED	REPLACE	VDC_TEXT
HPGL_SET_PEN_WIDTH	RESET_DEVICE	WORLD_COORDINATE_TEXT
HPGL_WRITE_BUFFER	R_ALLOC_OFFSCREEN	

Functions

Below is a list of all the Lisp functions for accessing the Starbase library functions. It gives the name of the functions and the types of arguments they require. See the Starbase documentation for information on what each function does. When an argument is a symbol, the value cell of the symbol is used. For more information see the chapter “Calling Non-Lisp Routines” in the *Lisp Programmer’s Guide*.

(await_event fixnum float symbol symbol)

(await_retrace fixnum)

(background_color_index fixnum fixnum)

(background_color fixnum float float float)

(block_move fixnum float float fixnum fixnum float float)

(dcblock_move fixnum fixnum fixnum fixnum fixnum fixnum fixnum)

(block_read fixnum float float fixnum fixnum simple-string fixnum)

(dcblock_read fixnum fixnum fixnum fixnum fixnum simple-string fixnum)

(block_write fixnum float float fixnum fixnum simple-string fixnum)

(dcblock_write fixnum fixnum fixnum fixnum fixnum simple-string fixnum)


(buffer_mode fixnum fixnum)

(character_expansion_factor fixnum float)


(character_height fixnum float)

(dccharacter_height fixnum fixnum)

(character_slant *fixnum float*)
 (character_width *fixnum float*)
 (dccharacter_width *fixnum fixnum*)
 (clear_control *fixnum fixnum*)
 (clear_view_surface *fixnum*)
 (clip_depth *fixnum float float*)
 (clip_indicator *fixnum fixnum*)
 (clip_rectangle *fixnum float float float float*)
 (concat_matrix *4x4-float-array 4x4-float-array 4x4-float-array*)
 (concat_transformation2d *fixnum 3x2-float-array fixnum fixnum*)
 (concat_transformation3d *fixnum 4x4-float-array fixnum fixnum*)
 (dc_to_vdc *fixnum fixnum fixnum fixnum symbol symbol symbol*)
 (define_color_table *fixnum fixnum fixnum 2d-float-array*)
 (define_raster_echo *fixnum
 fixnum
 fixnum
 fixnum
 fixnum
 fixnum
 simple-string*)
 (depth_indicator *fixnum fixnum fixnum*)
 (designate_character_set *fixnum simple-string fixnum*)
 (disable_events *fixnum fixnum fixnum*)
 (display_enable *fixnum fixnum*)
 (draw2d *fixnum float float*)
 (draw3d *fixnum float float float*)
 (dcdraw *fixnum fixnum fixnum*)
 (drawing_mode *fixnum fixnum*)




(echo_type *fixnum fixnum fixnum float float float*)
(dcecho_type *fixnum fixnum fixnum fixnum fixnum*)
(echo_update *fixnum fixnum float float float*)
(dcecho_update *fixnum fixnum fixnum fixnum*)
(enable_events *fixnum fixnum fixnum*)
(fill_color_index *fixnum fixnum*)
(fill_color *fixnum float float float*)
(fill_dither *fixnum fixnum*)
(flush_matrices *fixnum*)
(gclose *fixnum*)



To use `gerr_procedure`, you must write two C functions: one that is the error handler, and another that sets up the error handler by calling `gerr_procedure`. Load this C code with `extn:load-ofile`, and use `extn:defexternal` to define a Lisp version of the C function that calls `gerr_procedure`. Calling the Lisp version will set up the error handler.

(gerr_defaults)
(gerr_print_control *fixnum*)
(gerr_message *fixnum*)
(gescape *fixnum fixnum 64-element-fixnum-array 64-element-fixnum-array*)

If you want to call `gescape` with another type of array, you must use the non-Lisp function calling facility to define your own version. See “Calling Non-Lisp Routines” in the *Lisp Programmer’s Guide*.



(gopen *simple-string fixnum simple-string fixnum*)
(initiate_request *fixnum fixnum fixnum symbol*)
(inquire_color_table *fixnum fixnum fixnum 2d-float-array*)
(inquire_gerror *symbol symbol*)
(inquire_id *fixnum symbol simple-string symbol*)

(*inquire_input_capabilities* *fixnum*
symbol
symbol
symbol
symbol
symbol
symbol)

(*inquire_request_status* *fixnum* *fixnum* *fixnum* *symbol*)

(*inquire_sizes* *fixnum*
2x3-float-array
three-element-float-vector
three-element-float-vector
three-element-float-vector
symbol)

(*inquire_text_extent* *fixnum* *simple-string* *fixnum* *twelve-element-float-vector*)

(*interior_style* *fixnum* *fixnum* *fixnum*)

(*intra_character_space* *fixnum* *float*)

(*line_color_index* *fixnum* *fixnum*)

(*line_color* *fixnum* *float* *float* *float*)

(*line_repeat_length* *fixnum* *float*)

(*line_type* *fixnum* *fixnum*)

(*make_picture_current* *fixnum*)

(*mapping_mode* *fixnum* *fixnum*)

(*marker_color_index* *fixnum* *fixnum*)

(*marker_color* *fixnum* *float* *float* *float*)

(*marker_orientation* *fixnum* *float* *float*)

(*marker_size* *fixnum* *float* *fixnum*)

(*dcmarker_size* *fixnum* *fixnum*)

(*marker_type* *fixnum* *fixnum*)

(*move2d* *fixnum* *float* *float*)

(move3d *fixnum float float float*)
(dcmove *fixnum fixnum fixnum*)
(partial_polygon2d *fixnum float-vector fixnum fixnum fixnum*)
(partial_polygon3d *fixnum float-vector fixnum fixnum fixnum*)
(dcpartial_polygon *fixnum fixnum-vector fixnum fixnum fixnum*)
(perimeter_color_index *fixnum fixnum*)
(perimeter_color *fixnum float float float*)
(perimeter_repeat_length *fixnum float*)
(perimeter_type *fixnum fixnum*)
(polygon2d *fixnum float-vector fixnum fixnum*)
(polygon3d *fixnum float-vector fixnum fixnum*)
(dcpolygon *fixnum fixnum-vector fixnum fixnum*)
(polyline2d *fixnum float-vector fixnum fixnum*)
(polyline3d *fixnum float-vector fixnum fixnum*)
(dcpolyline *fixnum fixnum-vector fixnum fixnum*)
(polymarker2d *fixnum float-vector fixnum fixnum*)
(polymarker3d *fixnum float-vector fixnum fixnum*)
(dcpolymarker *fixnum fixnum-vector fixnum fixnum*)
(pop_matrix *fixnum*)
(pop_matrix2d *fixnum 3x2-float-array*)
(pop_matrix3d *fixnum 4x4-float-array*)
(push_matrix2d *fixnum 3x2-float-array*)
(push_matrix3d *fixnum 4x4-float-array*)
(push_vdc_matrix *fixnum*)
(read_choice_event *fixnum symbol symbol symbol symbol symbol*)

(read_locator_event fixnum symbol symbol symbol symbol symbol symbol symbol)
(rectangle fixnum float float float float)
(drectangle fixnum fixnum fixnum fixnum fixnum)
(replace_matrix2d fixnum 3x2-float-array)
(replace_matrix3d fixnum 4x4-float-array)
(request_choice fixnum fixnum float symbol symbol)
(request_locator fixnum fixnum float symbol symbol symbol symbol)
(sample_choice fixnum fixnum symbol symbol)
(sample_locator fixnum fixnum symbol symbol symbol symbol)
(set_locator fixnum fixnum float float float)
(set_p1_p2 fixnum fixnum float float float float float float)
(set_signals fixnum fixnum)
(append_text fixnum simple-string fixnum fixnum)
(text2d fixnum float float simple-string fixnum fixnum)
(text3d fixnum float float float simple-string fixnum fixnum)
(dctext fixnum fixnum fixnum simple-string)
(text_alignment fixnum fixnum fixnum float float)
(text_color_index fixnum fixnum)
(text_color fixnum float float float)
(text_font_index fixnum fixnum)
(text_line_path fixnum fixnum)
(text_line_space fixnum float)
(text_orientation2d fixnum float float float float)
(text_orientation3d fixnum float float float float float float)
(text_path fixnum fixnum)

(text_precision *fixnum fixnum*)

(text_switching_mode *fixnum fixnum*)

(track *fixnum fixnum fixnum*)

(track_off *fixnum*)

(transform_points *fixnum float-array float-array fixnum fixnum*)

(vdc_extent *fixnum float float float float float float*)

(vdc_to_dc *fixnum float float float symbol symbol symbol*)

(vdc_to_wc *fixnum float float float symbol symbol symbol*)

(viewport_justification *fixnum float float*)

(wc_to_vdc *fixnum float float float symbol symbol symbol*)

(write_enable *fixnum fixnum*)

Device I/O Functions

The Lisp access functions for the Device I/O library (DIL) are defined in the `hp-ux_3i` module, which can be conditionally loaded with

```
(require "hp-ux_3i")
```

All of the symbols described in this section are interned in the `hp-ux_3i` package.

NOTE

The `hp-ux_3i` module should not be loaded or used when either the `hp-ux_3w` or `hp-ux_3g` module is loaded.

Constants

The following useful constants are defined in the `hp-ux_3i` module.

```
HPIBWRITE  
HPIBREAD  
HPIBATN  
HPIBEOI  
HPIBCHAR
```

Structure Arguments

The device I/O function `hpib_io` takes as one of its arguments an array of C structures. To facilitate calling `hpib_io` from Lisp, we have defined an equivalent Lisp structure called `iodetail`. The definition of this structure is

```
(defstruct iodetail  
  (mode      0 :type unsigned-byte)  
  (terminator 0 :type unsigned-byte)  
  (count     0 :type fixnum)  
  (buf       "" :type simple-string))
```

Since `hpib_io` actually takes an array of such structures, the function `make-iodetail-array` is also defined.

```
(hp-ux_3i:make-iodetail-array n)
```

Function

A call to this function returns an *n*-element one-dimensional array of `iodetail` structures which have been initialized to the default values (see definition above). The second argument to the Lisp version of `hpib_io` should be an array returned by `make-iodetail-array`.

Functions

This is a list of all the Lisp functions for accessing the Device I/O Library (DIL) functions. It gives the name of the functions and the types of parameters they require. See section three of the *HP-UX Reference* for information on what each function does.

(*gpio_get_status fixnum*)

(*gpio_set_ctl fixnum fixnum*)

(*hpib_abort fixnum*)

(*hpib_bus_status fixnum fixnum*)

(*hpib_card_ppoll_resp fixnum fixnum*)

(*hpib_eoi_ctl fixnum fixnum*)

(*hpib_io fixnum iodetail-array fixnum*) ; See description of iodetail
; structure above

(*hpib_pass_ctl fixnum fixnum*)

(*hpib_ppoll fixnum*)

(*hpib_ppoll_resp_ctl fixnum fixnum*)

(*hpib_ren_ctl fixnum fixnum*)

(*hpib_rqst_srvce fixnum fixnum*)

(*hpib_send_cmd fixnum simple-string fixnum*)

(*hpib_spoll fixnum fixnum*)

(*hpib_status_wait fixnum fixnum*)

(*hpib_wait_on_ppoll fixnum fixnum fixnum*)

In the current release, calling *io_burst* has no effect (it is a no-op). This is provided so that code written for this release need not be changed for future versions that support *io_burst*.

(*io.eol_ctl fixnum fixnum fixnum*)

(*io_get_term_reason fixnum*)

(*io_reset fixnum*)

`(io_speed_ctl fixnum fixnum)`

`(io_timeout_ctl fixnum fixnum)`

`(io_width_ctl fixnum fixnum)`

Networking Functions

The Lisp access functions for Local Area Networking (LAN) are defined in the `hp-ux_3n` module, which can be conditionally loaded with

`(require "hp-ux_3n")`

All of the symbols described here are interned in the `hp-ux_3n` package. For the meaning of these functions see the reference pages in your LAN documentation.

`(errnet)` ; Returns the value of the `errnet` variable

`(netunam simple-string simple-string)`

`(net_aton simple-string simple-string fixnum)`

`(net_ntoa simple-string simple-string fixnum)`

Chapter 5

User I/O

Introduction

This chapter explains how to use the Common Lisp I/O functions to input from and output to various types of files. Most of the information presented applies to I/O for functions that will be running independent of NMODE, but a few functions provided by NMODE for user input and output are discussed.

Prerequisites

Since the Common Lisp system runs on top of HP-UX, the behavior of the I/O functions is sometimes dependent on factors controlled by HP-UX. In particular, input and output to and from a terminal's special device file (of which windows are a special case) are affected by the `termio` structure associated with that device file. This chapter attempts to give you all the information you need to deal with these things; for details see the `ioctl(2)` and `termio(4)` entries in the *HP-UX Reference*.

This chapter assumes that you have a basic understanding of Windows/9000 concepts, such as the difference between graphics and `term0` windows. This information is covered in the *HP Windows/9000 User's Manual*, and the *HP Windows/9000 Programmer's Manual*.

Organization

The first part of this chapter deals with I/O from/to files. First, text files are discussed, then device special files. The device file section covers `term0` window I/O, and I/O with graphics windows. Next there is a brief section that suggests a way of getting input from the mouse. The last section deals with user I/O in the NMODE programming environment.

File I/O

Text Files

Text files are “real” files: collections of characters stored on disk somewhere in the HP-UX file hierarchy, like `/usr/include/window.h` or `/etc/passwd`. Using Lisp to read from or write to such a file is easy. Just use one of the Lisp functions `open` or `with-open-file` to bind a stream to the desired file and call any Lisp I/O function that takes a stream argument. When used with text files, all the Common Lisp I/O functions behave as described in Steele.

If you have HP’s local area network (LAN) hardware and software, you can even do I/O on a file on another computer’s file system via RFA (Remote File Access). The upcoming example demonstrates doing this conditionally.

Example

Suppose you are an R&D manager trying to keep an eye on your staff. You’re worried that the programmers are not documenting their code sufficiently, so you write the following program to help you analyze the situation. It takes an output file name, an input file name, and some optional arguments that are used for accessing files via LAN (if desired). The input file is read, and any line containing a semicolon is written to the output file. When processing is complete, a summary is written to `*standard-output*`.

```
(require "hp-ux_3n")

(defun analyze-comments (out-file in-file &optional
                        (rem-system nil) (password ""))
  (let ((real-in-file (get-real-file-name in-file rem-system password)))

    (with-open-file (outstream out-file :direction :output)
      (let ((line-count 0)
            (comment-count 0))
        (with-open-file (instream real-in-file :direction :input)
          (do ((line (read-line instream nil 'eof)
                    (read-line instream nil 'eof)))
              ((eq line 'eof))
              (incf line-count)
              (when (commentp line)
                (incf comment-count)
                (write-line line outstream)
              )
            ))
          (format outstream "Total Lines: ~A~%Comment Lines: ~A~% Percentage:
~4,1F~%"
                  line-count comment-count (* 100 (/ comment-count line-count)))
        )
    )
  )
```

```

)
)

(defun get-real-file-name (file-name system password)
  (cond ((simple-string-p system)
        ;; Try to set up access to remote system
        (if (< (hp-ux_3n:netunam system password) 0)
            (error "Invalid password specified for ~A" system))
        (concatenate 'simple-string system file-name))
        (t file-name)
    )
)

;; Currently stupid, returns true even for ;'s imbedded in strings
(defun commentp (str)
  (find #\; str :test #'char=)
)

;; An example call
(analyze-comments "results" "/users/stud/lispfiles/foo.1" "/net/stud"
                  "root:zanziBar")

```

Terminal Device Files

Terminal device (or special) files are HP-UX device files that correspond to some kind of terminal. Windows have device files that behave like a terminal device, so they are included in this category. These files, such as `/dev/console` and the files in `$WMDIR`, can be opened with `open` or `with-open-file` and read from or written to with the Common Lisp I/O functions.

Before you do any input or output from/to a terminal device, you must make sure that the `termio` structure for that device is set correctly. The `termio` structure contains fields that control things like how carriage returns are treated, what the backspace character is, and whether or not characters are automatically echoed. See *termio(4)* in the *HP-UX Reference* for a description of all the available fields. If you're using a device file that does not correspond to a window created by your program, then you also need to make sure that your application resets the `termio` structure to whatever it was before your program took over. To modify a device file's `termio` structure you need to have an HP-UX file descriptor for the file. This means that you will have to call the HP-UX system function *open(2)* (`hp-ux_2:open`) before using the Lisp function `open`.

Sequence of Events

The basic steps in a program that performs user I/O are:

1. Get a file descriptor corresponding to the device you want to use by calling `hp-ux_2:open` (not necessary if you are using standard input and/or standard output since their file descriptors are 0 and 1 respectively). If you create the window yourself you will have the file descriptor already because the utilities to establish communication with a window open the file and return the file descriptor.
2. If you are not doing I/O to a window that your program created, save the device's termio structure.
3. Set the termio structure to the desired values.
4. Call Common Lisp `open` (or `with-open-file`) with the HP-UX pathname of the device you are using (for instance `/dev/screen/myapp`). You should open separate streams for input and output.
5. Do whatever your application is supposed to do, doing reads and writes from the streams you opened.
6. Close the input and output streams (with Lisp `close`).
7. Restore the device's termio structure to its saved state, if necessary.
8. Call `hp-ux_2:close` with the device's HP-UX file descriptor. For windows, this is done by the utilities for terminating window communication.

Setting the Termio Structure

A terminal device's termio structure is set by calling the HP-UX system function `ioctl(2)`. The `hp-ux_2` module defines a function `tty-ioctl` and a Lisp termio structure to simplify calling `ioctl`.

There are four basic operations that you will want to have available:

1. Save a termio structure.
2. Set up termio for output and canonical input.
3. Set up termio for output and raw input.
4. Restore a termio structure.

The Lisp code to perform these four operations will be described shortly. First, let's look at two types of input.

Canonical Input

Canonical input lets the operating system do much of the dirty work of keyboard input. When canonical processing is enabled, keyboard input is processed in units of lines. You can read the input from Lisp in characters or lines, but your application will not "see" any part of a line typed by the user until they press the return key. Echoing and backspacing are handled by the operating system.

Raw Input

Raw input is the opposite of canonical input. Characters are passed through to your application as they are typed and are not echoed to the screen. Some special keys are handled by HP-UX. For instance, Signals are still sent for the characters set in the INTR and QUIT slots of termio's c_cc array. You must handle echoing and backspacing yourself.

Sample Code

The following Lisp code defines four useful functions for dealing with termio structures that are used in the examples in this chapter. This source is in the file \$LISP/doc/examples/termio.l.

```
.....
;
; File:          termio.l
; SCCS:         @(#) $hi/doc/examples/termio.l 1.1@(#) 4/23/86 13:53:03
; Description:   Stuff for dealing with a file's termio structure
; Language:     Lisp
; Package:      USER
;
; (c) Copyright 1986, Hewlett-Packard Company, all rights reserved.
;
.....
(provide "termio")
(in-package 'user)

(require "hp-ux_2")

;;;
;;; Defconstants for a few of the things found in termio.h and fcntl.h
;;;
(defconstant  VINTR    0)
(defconstant  VQUIT   1)
(defconstant  VERASE  2)
(defconstant  VKILL   3)
(defconstant  VEOF    4)
(defconstant  VEOL    5)
(defconstant  VMIN    4)
(defconstant  VTIME   5)
(defconstant  ICRNL   #o0000400)
(defconstant  BRKINT  #o0000002)
(defconstant  IGNPAR  #o0000004)
```

```

(defconstant IXON #o0002000)
(defconstant OPOST #o0000001)
(defconstant ONLCR #o0000004)
(defconstant TABO 0)
(defconstant ISIG #o0000001)
(defconstant ICANON #o0000002)
(defconstant XCASE #o0000004)
(defconstant ECHO #o0000010)
(defconstant ECHOE #o0000020)
(defconstant ECHOK #o0000040)
(defconstant B9600 #o0000020)
(defconstant CS8 #o0000140)
(defconstant CREAD #o0000400)
(defconstant CLOCAL #o0010000)

```

```

(defconstant O_NDELAY #o04)
(defconstant F_GETFL 3)
(defconstant F_SETFL 4)

```

```

;;;
;;; Set-canonical sets the terminal device to do canonical input
;;; with carriage returns handled in a way appropriate for Common
;;; Lisp I/O functions.
;;;

```

```

(defun set-canonical (fd)
  (let ((tio (hp-ux_2:make-termio))
        file-control-flag
        )
    ;; Get the current termio structure
    (hp-ux_2:tty-ioctl fd hp-ux_2:TCGETA tio)
    ;; Change the appropriate fields
    (setf (aref (hp-ux_2:termio-c_cc tio) VINTR) (code-char 3))
    (setf (aref (hp-ux_2:termio-c_cc tio) VQUIT) (code-char 28))
    (setf (aref (hp-ux_2:termio-c_cc tio) VERASE) (code-char 8))
    (setf (aref (hp-ux_2:termio-c_cc tio) VKILL) (code-char 21))
    (setf (aref (hp-ux_2:termio-c_cc tio) VEOF) (code-char 4))
    (setf (aref (hp-ux_2:termio-c_cc tio) VEOL) (code-char 0))
    (setf (hp-ux_2:termio-c_lflag tio) (logior (hp-ux_2:termio-c_lflag tio)
                                              ECHO
                                              ICANON
                                              ISIG
                                              ECHOK
                                              ECHOE))
    (setf (hp-ux_2:termio-c_iflag tio) (logior (hp-ux_2:termio-c_iflag tio)
                                              BRKINT
                                              IGNPAR
                                              ICRNL
                                              IXON))
    (setf (hp-ux_2:termio-c_oflag tio) (logior (hp-ux_2:termio-c_oflag tio)

```

```

                                OPOST
                                ONLCR
                                TABO))
(setf (hp-ux_2:termio-c_cflag tio) (logior (hp-ux_2:termio-c_cflag tio)
                                B9600
                                CS8
                                CREAD
                                CLOCAL))

;; Put the new termio values into effect
(hp-ux_2:tty-ioctl fd hp-ux_2:TCSETA tio)
;; Get the current file control value and turn off O_NDELAY
(setf file-control-flag (logand (lognot O_NDELAY) (hp-ux_2:fcntl fd F_GETFL
0)))
(hp-ux_2:fcntl fd F_SETFL file-control-flag)
)
)

;;;
;;; Set-raw sets the terminal device to return characters as they are typed
;;; without echoing them. In this case, the user will have to handle
;;; echoing, backspacing, etc. Carriage returns are handled correctly.
;;;
(defun set-raw (fd)
  (let ((tio (hp-ux_2:make-termio)))
    ;; Get the current termio structure
    (hp-ux_2:tty-ioctl fd hp-ux_2:TCGETA tio)
    ;; Change the appropriate fields
    (setf (hp-ux_2:termio-c_lflag tio) (logand (hp-ux_2:termio-c_lflag tio)
                                                (lognot (logior ECHO ICANON
XCASE))))
    (setf (hp-ux_2:termio-c_lflag tio) (logior (hp-ux_2:termio-c_lflag tio)
                                                ISIG))
    (setf (aref (hp-ux_2:termio-c_cc tio) VMIN) (code-char 1))
    (setf (aref (hp-ux_2:termio-c_cc tio) VTIME) (code-char 0))
    (setf (hp-ux_2:termio-c_iflag tio) (logior (hp-ux_2:termio-c_iflag tio)
                                                ICRNL))
    (setf (hp-ux_2:termio-c_oflag tio) (logior (hp-ux_2:termio-c_oflag tio)
                                                OPOST
                                                ONLCR))

    ;; Put the new termio values into effect
    (hp-ux_2:tty-ioctl fd hp-ux_2:TCSETA tio)
  )
)

(defun (defvar *save-termio-list* nil) ; Association list of file descriptors and
termios

;;;
;;; Save-termio saves a terminal device's termio structure on *save-termio-list*
;;;

```

```

(defun save-termio (fd)
  (let ((tio (hp-ux_2:make-termio)))
    (when (assoc fd *save-termio-list* :test #'=)
      (error "Overwrites currently saved one"
             "Termio for file descriptor ~A already saved." fd))
    ;; Get the current termio structure and save it
    (if (hp-ux_2:tty-ioctl fd hp-ux_2:TCGETA tio)
        (setf *save-termio-list* (acons fd tio *save-termio-list*)))
    )
  )

;;;
;;; Reset-termio restores a terminal device's termio structure from
;;; *save-termio-list*.
;;;
(defun reset-termio (fd)
  (let* ((assoc-pair (assoc fd *save-termio-list* :test #'=))
         (tio (cdr assoc-pair))
        )
    (unless tio
      (error "No termio structure saved for file descriptor ~A" fd))
    (setf *save-termio-list* (delete assoc-pair *save-termio-list*))
    (hp-ux_2:tty-ioctl fd hp-ux_2:TCSETA tio)
  )
)

```

Term0 Windows

Term0 windows emulate a terminal. The default window that you get when you use the *wsh* command is a term0 window. Term0 windows are appropriate where display performance is not as much of a concern as simplicity of use. Graphics windows are faster but there is no echoing in them, so you must do your own echoing (if you want it) in raw mode with fast alpha or font manager routines.

Window-Smart and Window-Dumb

Applications can be either window-smart or window-dumb. A window-smart program is window system dependent. It may call window library routines and/or recognize window system signals. A window-dumb application does not "know" anything about the window system. It could be run at a terminal. Window-dumb applications are usually run in term0 windows.

This section shows the code for two applications that use term0 windows. One of them is window-smart; it creates a term0 window solely for use by the application. The second is window-dumb; it uses whatever window (or terminal) it was invoked from. Functions that create their own windows for I/O can be called directly from within NMODE. For window-dumb applications, you can get to a Lisp listener from an NMODE buffer in Lisp mode with **Lisp-L**, and then call your function from there.

Window-Smart Example

Here is the code for a simple function called *query-user*. It creates a window, prints a prompt (supplied by the caller), and returns whatever the user types before a carriage return. The overhead of creating the window is hardly worth the trouble, but hey, it's only an example.

```
(require "windowutil" "$LISP/doc/examples/windowutil")
(require "termio" "$LISP/doc/examples/termio")

(in-package 'user :use '(hp-ux_3w windowutil))

(defun query-user (wm-fildes window-name prompt)
  (let ((window-path (make-string 25))
        t0-fildes
        iostream
        reply)
    (wmpathmake "WMDIR" window-name window-path)

    ;; Create Window
    (wcreate_term0 wm-fildes window-path 200 100 80 4 80 4 80 24
                  "/usr/lib/raster/8x16/lp.8U"
                  "/usr/lib/raster/8x16/lp.b.8U"
                  2
```

1)

```
(unwind-protect
  (and
    (setq t0-fildes (establish-t0-communication window-path))
    (set-canonical t0-fildes)
    (setq iostream (open window-path :direction :io :if-exists :overwrite))
    (wtop t0-fildes SETTOP)
    (wselect t0-fildes 1)
    (write-line prompt iostream)
    (setq reply (read-line iostream))
  )
  ;; Protect Forms
  (if iostream (close iostream))
  (if t0-fildes (terminate-t0-communication t0-fildes))
  (wdestroy wm-fildes window-path))
reply)
)

;; Test
(setq wmfid (establish-wm-communication))
(query-user wmfid "Jimi" "Hey Joe, where you going with that pizza in your
hands?")
(terminate-wm-communications wmfid)
```

In a practical application one would not create a window to ask the user only one question. However, you could create a window at the beginning of the application that would be dedicated to querying the user and then expose it and write to it as necessary.

Window-Dumb Example

Window-dumb applications use the file descriptor for standard input (1) as the argument to the functions that set the termio structure. They use the Common Lisp streams `*standard-input*` and `*standard-output*` for stream I/O. These streams are the defaults for many I/O functions, so it is not always necessary to mention them explicitly.

Window-dumb applications will be run from a read-eval-print loop running in a `term0` window. Here is the code for a simple window-dumb application. The program reads numbers entered by the user until the symbol `q` is entered. It then prints a message and the sum of the numbers.

```
(require "termio" "$LISP/doc/examples/termio")
(in-package 'user)

(defun simple-adder ()
  (save-termio 0) ; Standard Input's file descriptor is 0
  (set-canonical 0)
  (unwind-protect
```

```

(do ((sum 0)
    (n 0))
  ((eq n 'q) (format t "The sum of the numbers entered is ~A~%" sum))
  (format t "Enter number: ") ; t means use *standard-output*
  (setq n (read))           ; Uses *standard-input*
  (if (numberp n)
      (incf sum n)
    )
  ;; Protect Forms
  (reset-termio 0)
)
)

```

Graphics Windows

You can use graphics windows for text-based applications, but they require a little more work. Because they are graphics devices, graphics windows do not echo characters as they are typed at the keyboard. The application writer (you) must echo characters using Windows/9000 fast alpha or font manager routines. This means you need to get each character as it is typed, so raw input must be used.

Example

Getting input from a graphics window can be simple if you're only concerned with getting a character or two of input without echoing them. Consider the method `:user-y-or-n-p` in the tic-tac-toe program. It would be reasonable to make it so that the player would just type a y or an n at the keyboard rather than use the mouse. This would make it advantageous to have an instance variable whose value would be the input stream for the window. This opening of the stream would be in the `:init` method, along with a call to `set-raw`. Only routines that would need to be changed are shown here. Changed lines are marked with `**`.

```

;;;
;;; The ttt-game instance type maintains all the info needed to communicate
;;; with the game window. Its methods run the game.
;;;
(define-type ttt-game
  (:var board) ; An instance of ttt-board
  (:var window-path) ; The HP-UX pathname of the window special
device
  (:var wm-fildes) ; File descriptor of window manager interface
  (:var gr-fildes) ; File descriptor of graphics window
  (:var fa-fildes) ; File descriptor of fast-alpha interface
  (:var input-stream) ; Stream opened to the window device **
  (:var marker-font-id)
  (:var prompt-font-id)
  (:var x-player-move) ; Method to generate x's move

```



```

        (:var o-player-move) ; Method to generate o's move
        :all-initable)

;;;
;;; Initialize the game. This method does all the work of
;;; setting up the window interface.
;;;
(define-method (ttt-game :init) (keylist)
  (setq wm-fildes (establish-wm-communication))
  (let ((temp (make-string 25)))
    (wmpathmake "WMDIR" "ttt" temp)
    (setq window-path temp))
  ;; Create Window
  (wcreate_graphics wm-fildes
                   window-path
                   700
                   100
                   270
                   330
                   270
                   330
                   SETRETAIN
                   SETNOBANNER)
  (setq gr-fildes (establish-gr-communication wm-fildes window-path))
  (setq fa-fildes (establish-fa-communication wm-fildes window-path))
  (set-raw fa-fildes) ; ** set-raw would have to be defined
  (setq input-stream (open window-path :direction :input)) ; **
  (setq marker-font-id (fafontload fa-fildes "/usr/lib/raster/18x30/pica.8U"))
  (setq prompt-font-id (fafontload fa-fildes "/usr/lib/raster/7x10/lp.8U"))
  )

;;;
;;; :User-y-or-n-p returns true if the user types a y (or Y), nil otherwise.
;;;
(define-method (ttt-game :user-y-or-n-p) ()
  (wselect gr-fildes 1) ; **
  (char= (read-char input-stream) #\y) ; **
  )

;;;
;;; :Cleanup "undoes" all the game's connections to Windows/9000
;;;
(define-method (ttt-game :cleanup) ()
  (close input-stream) ; **
  (fafontremove fa-fildes marker-font-id)
  (fafontremove fa-fildes prompt-font-id)
  (terminate-fa-communication fa-fildes)
  (terminate-gr-communication gr-fildes)
  (wdestroy wm-fildes window-path)
  (terminate-wm-communication wm-fildes)

```

)

The Mouse

The recommended method for using the mouse as a locator device is to poll using `hp-ux_3w:wgetlocator`. This is the way moves are input in the tic-tac-toe program (see the `:get-user-move` method). If you want to use the mouse with pop-up menus, see the “Pop-Up Menus” section below.

Pop-Up Menus

The Windows/9000 library provides some functions for creating and using pop-up menus. These functions are described in the *HP Windows/9000 Reference* and the *HP Windows/9000 Programmer's Manual*, documents written primarily for people programming in C. Consequently, this section has been written to show you how to create and use Windows/9000 pop-up menus from Lisp. Since you will still be calling the Windows/9000 functions, you should have the Windows/9000 reference manual handy while you read this.

The function and macro described in this section are only one way of using the Windows/9000 menu capabilities from Lisp. It does not implement the full functionality available. You are encouraged to expand this example to suit your particular needs.

Example

The following code from the file `$LISP/doc/examples/windowutil.1` implements a simple pop-up menu facility.

Popping Up a Menu

The primary function is `make-and-activate-menu`, which creates a menu, displays it at the current locator position, and returns the user's selection. It takes the following arguments:

- fd** The HP-UX file descriptor of the window the menu is to be associated with.
- name** A string giving the title of the menu. This will be a non-selectable item at the top of the menu.

menu-items A list that specifies the contents of the menu in order. An element in this list can be

- The keyword `:line`. This specifies a non-selectable horizontal line across the menu.
- A string to be displayed as a non-selectable item.
- A list. This specifies a selectable item. The first element of the list must be a string to be displayed for the item. The rest of the list can be whatever you want; the entire list is returned if the item is selected by the user. Useful things to put as the rest of the list are discussed shortly.

If the user selects a selectable menu item, `make-and-activate-menu` returns the list that represents that item, otherwise it returns `nil`. Here is the definition of `make-and-activate-menu`.

```
(defun make-and-activate-menu (fd name menu-items)
  (let ((menu-id (wmenu_create fd MENU_POPUP -1 MENU_NOPARENT MENU_NOPARENT))
        selected-menu-id
        selected-menu-item
      )
    (declare (special selected-menu-id selected-menu-item))
    (make-menu-item fd menu-id name :no-select :no-tracking)
    (make-menu-item fd menu-id :line)
    (dolist (item menu-items)
      (cond ((listp item)
             (make-menu-item fd menu-id (first item)))
            ((eq item :line)
             (make-menu-item fd menu-id :line))
            ((stringp item)
             (make-menu-item fd menu-id item :no-select))
            (t
             (cerror "Ignores item" "Invalid pop-up menu item ~A" item))
            ))
      (wmenu_activate fd menu-id MENU_ACT_IM)
      (wmenu_eventread fd 'selected-menu-id 'selected-menu-item)
      (wmenu_delete fd menu-id)
      (if (>= selected-menu-item 0)
          (nth (- selected-menu-item 2) menu-items)
          nil
        )
      )
    )
  )
```

Creating Menu Items

In `make-and-activate-menu`, menu item creation is handled by the `make-menu-item` macro. A call to this macro expands into a call to the Windows/9000 function `wmenu_item`.

The required arguments to `make-menu-item` are the HP-UX file descriptor of the window to be associated with the menu, the menu id of the menu, and the string to be displayed for the item (or `:line` to create a separator item). Any remaining arguments are used to specify the selectability, display color, and tracking characteristics of the item being created. By default, the created item is selectable, displayed in normal text, and is tracked in inverse-video. To override these defaults, include one or more of these keywords as arguments (note that these are not `&key` arguments).

`:no-select` Make the item non-selectable.

`:grey` Display the item in "shaded" grey text.

`:no-tracking` Make it so that there is no inverse-video highlight displayed when the locator is over that item in the menu.

Here is the source to the macro `make-menu-item`. Remember that even though it is discussed here after `make-and-activate-menu`, in the source file, the macro definition must come first, so that it is available when `make-and-activate-menu` is defined.

```
(defmacro make-menu-item (fd menu-id item-string &rest traits)
  (if (eq item-string :line)
      '(hp-ux-return
        (wmenu_item ,fd ,menu-id MENU_NEWITEM MENU_SEPARATOR MENU_NOTSELECTABLE
          ""))
      ;; Else
      (let* ((select-mask (if (member :no-select traits) MENU_NOTSELECTABLE
                              MENU_SELECTABLE))
             (display-mask (if (member :grey traits) MENU_DISPGREY
                                MENU_DISPORM))
             (track-mask (if (member :no-tracking traits) MENU_TRACKNOCHNG
                              MENU_TRACKINV))
             (mask (logior select-mask display-mask track-mask)))
        '(hp-ux-return
          (wmenu_item ,fd ,menu-id MENU_NEWITEM MENU_STRING ,mask ,item-string))
        )
      )
  )
```

Using make-and-activate-menu

In the tic-tac-toe game program, the `:user-y-or-n-p` method of the `ttt-game` instance type uses `make-and-activate-menu` to get a yes or no answer from the user.

```
(define-method (ttt-game :user-y-or-n-p) ()
  (wsetlocator gr-fildes -100 100)
  (second (make-and-activate-menu gr-fildes "Yes or No"
    '(("Yes" yes) ("No" nil))))
)
```

Notice that it treats a menu cancellation, or selection of a non-selectable item as a "no" answer.

This example uses very simple lists for its selectable menu items. Some ideas for useful menu item lists are:

- The second element of the list is a parameterless function to be funcalled when that menu item is selected. For instance,
(`Create File` `#'create-file-command`).
- The second element of the list is a list representing another pop-up menu item list to be selected from. For instance,
(`Create >>` ((`"File"` `#'create-file-command`) (`"Directory"` `#'create-dir-command`)))


Ideas for Expansion

The two routines discussed here are designed only to give you a platform from which to develop a more detailed menu facility. This section mentions a few capabilities you might want to implement.

With the current facility, there is no way to create a menu, save it, and then pop it up whenever you choose; the menu is created every time. A menu data structure could be created (with instances or structures). You could then split `make-and-activate-menu` into separate creation and activation functions.

If you implement this suggestion, then you may also want to extend `make-menu-item` to allow modifying existing menu items. Your call to `hp-ux_3g:wmenu_item` would then need to specify the item id of the item, instead of `MENU_NEWITEM`.


NMODE I/O



The internal structure of the NMODE environment is not currently documented for customers because any programs written to use that information may become useless for future versions of this product. This means that you'll only be able to integrate relatively simple (in terms of I/O) programs into the environment.

When you are running NMODE and you call a Lisp input function such as `read-line` with `*standard-input*`, no input is available to that function until you execute the NMODE Execute Form Command (**Lisp-E**). Then the function reads starting at the beginning of the line that the cursor was on when you typed **Lisp-E**.

If you're creating an application function that will be called from within the NMODE environment, there are three basic approaches.


1. Have your function create its own window. In this case the information presented elsewhere in this chapter applies, and the program will behave the same when called from NMODE as when called from a simple read-eval-print loop.
 2. Have your function use the streams `*standard-input*` and `*standard-output*`. This situation is described in this section.
 3. Use a combination of NMODE supplied functions for input, and `*standard-output*` for output. This situation is described in this section.
- 

Standard Input and Output Streams

When NMODE is running, the variable `*standard-output*` is bound in such a way that any output sent to it will appear in the NMODE output buffer. This is the default destination for most of the Common Lisp output functions.

When NMODE is running, the variable `*standard-input*` is bound so that it is attached to the current buffer. However, no input is available until the user types the **Lisp-e** command. When that happens, whatever input function you called starts reading from the beginning of the line until it is satisfied. This makes `read-char` impractical since it always reads only the first character of the line.

One advantage to using the standard streams for programs is that they will work from NMODE or a simple read-eval-print loop.



NMODE I/O Functions

There are a few simple functions for displaying messages and getting a line of user input that are defined within the NMODE environment. These are useful for minor customization functions. These functions are in the `nmode` package.

`(nmode:prompt-for-string prompt-string default-string)` *Function*

When called, `prompt-for-string` displays *prompt-string* followed by (Default is: *default-string*). If an empty string is entered, the default string is returned, otherwise the string that was entered is returned. If *default-string* is `nil`, then no default is displayed in the prompt, and if the user just types Return, "" is returned.

`(nmode::write-prompt string)` *Function*
`(nmode:write-message string)` *Function*

Calling `write-prompt` displays its argument (a string) in the current NMODE screen's prompt area for approximately two seconds. The function `write-message` does the same, but the message remains on the screen until replaced by another call to `write-message`.

Example

Here's a function that provides access to Common Lisp function documentation strings from NMODE.

```
(defun nmode::function-documentation-command ()
  (let* ((function-name (nmode:prompt-for-string "Function Name: " nil))
         (function-sym (read (make-string-input-stream function-name) nil nil))
         (docstring (documentation function-sym 'function)))
    (if docstring
        (format t "~%-A%" docstring)
        (nmode::write-prompt
         (format nil "No documentation string for ~A" function-name)))
    )
  )
)
```

After definition, you could bind a key to this function with the forms

```
(in-package 'nmode)
(add-to-command-list 'lisp-command-list (x-char M-?)
 'function-documentation-command)
(nmode-establish-current-mode)
```

Chapter 6

Delivery

Introduction

Once you have written and tested an application, you need to be able to make it available to users. There are two basic scenarios for using an application written in Lisp.

- The user is also an NMODE user. In this case they only need to know what file(s) to load and the name of the main function.
- The user only wants to run your application. They don't know or care about Lisp and NMODE. In this case you will need to wrap up your application into a file that can be executed as an HP-UX command.

This chapter discusses several ways of making it possible to run a Lisp program as a command.

Dump Files

Dump files are executable files that “capture” a particular Lisp environment. The environment can then be restored by executing the file as an HP-UX command. When you bring up the NMODE you created with the `make-nmode` script, you are restoring a dump file. You can create dump files that will bring up whatever Lisp application you wish.

There are three basic steps to wrapping up an application into a dump file.

1. Bring up a read-eval-print loop by executing the `lisp-loader` command.
2. Load all the code required by your application.
3. Create a dump file of the Lisp system with the function `system:save-world`. In this call you will specify the forms you wish to have executed when the dump file is executed. One of these forms should call the main function of your application.

Example

This example shows how to create a dump file that, when restored, runs the tic-tac-toe program described earlier in this manual. Assuming that you've invoked Lisp with the HP-UX command

```
lisp-loader -t 4000000 -b .62 -m 200000
```

the following forms would be typed.

```
(in-package 'user)
;; Load the required modules
(require "hp-ux_3g")
(require "windowutil" "$LISP/doc/examples/windowutil")
(require "objects")
(load "$LISP/doc/examples/tictactoe")

;; Create dump file
(system:save-world "$LISP/bin/ttt-dump" '((ttt:tic-tac-toe)
                                           (system:exit))
                  "Entering Tic-Tac-Toe Game")
(system:exit)
```


After this, if \$LISP/bin is in your HP-UX PATH environment variable, you can run the tic-tac-toe game from a shell by typing

```
ttt-dump
```

Drawbacks

Note that the file is large (over two megabytes). If you have more than one or two Lisp applications that you want to run this way, the overhead in disc space is substantial. Consequently, you may want to consider the methods described under "General-Purpose Dump File".


General-Purpose Dump File



You can avoid the disc overhead of having one dump file for each application by creating a single dump file that contains the support code required by most of the applications you wish to run. Then you must implement a means of loading and executing the desired application when this dump file is restored. Two such techniques are described in this section, one that uses a Lisp function to load and execute the appropriate code based on command line arguments, and one that uses HP-UX shell scripts.

What you should load into the dump file you use to run your applications depends on the applications. Ideally you should only have to load things that are used by all your programs. At the same time, you don't want to have to load anything large at run time. You have to resolve the trade offs between time and space. The examples in this section assume that your applications require windowing, graphics, and objects support. Note that some applications may need more heap space than the examples allow (~ 3 megabytes).


Using Command Line Arguments



The function `sys:get-program-args` (an HP extension) returns a simple vector containing the strings that the user typed to invoke the Lisp process. You can use this capability to decide what program to load and run when a dump file is restored by having the user give the name of the desired application on the command line that restores the dump file. For instance, if your general-purpose dump file is named `run-lisp-app`, and the user wishes to run an application named `circuit-analyzer`, they would type this line to an HP-UX shell:

```
run-lisp-app circuit-analyzer
```

Somewhere in the initialization forms of `run-lisp-app` would be a call to a function that loads and executes a program based on the first command argument to `run-lisp-app`. This dispatching function would know what code to load, and how to invoke the top-level function of the circuit analyzer application.



Example Dispatching Function

Here is a definition of such a dispatching function:

```
(defun dispatch ()
  (let* ((args (sys:get-program-args))
        (appl-name (if (> (length args) 1) (svref args 1) nil))
        )
    (cond ((string= appl-name "ttt")
           (lisp:load "$tee/tictactoe")
           ;; The function must be funcalled because of the way
           ;; symbols are resolved at read time. This technique
           ;; delays resolution until execution time.
           (funcall (intern "TIC-TAC-TOE" (find-package 'ttt)))
           )
          ;; Add clauses for other applications here
          (t (format t "No known application ~A%" appl-name))
          )
    )
  )
)
```

This version of `dispatch` only knows about one application: the tic-tac-toe game. New applications would be added as similar clauses to the `cond` that load and execute the appropriate programs.

Here are the steps to create a general-purpose dump file that uses this dispatching mechanism. It assumes that `dispatch` is defined in either of the files `dispatch.l` or `dispatch.b`.

1. Execute the HP-UX command

```
lisp-loader -t 6500000 -b .55 -m 200000
```

2. Type in the following forms:

```
(in-package 'user)
(require "objects")
(require "hp-ux_2")
(require "hp-ux_3g")
(require "hp-ux_3w")
(require "windowutil" "$LISP/doc/examples/windowutil.b")
(system:save-world "$LISP/bin/run-lisp-app" '((load "dispatch")
                                             (dispatch)
                                             (sys:exit)))

(system:exit)
```

After this dump file is created and `$LISP/bin` has been added to their `PATH` environment variable, a user can run the tic-tac-toe program with the HP-UX command:

run-lisp-app ttt

Notice that the dispatching function is loaded when the dump file is restored. This allows you to change the definition of `dispatch` (to add a new application, for instance) without having to create a new dump file.

Using a Script

You can run a Lisp application by writing a script containing the forms you would execute to bring up the program if you were doing it by hand.

One of the drawbacks to this approach is that it is slower than using a dump file since things must be loaded at execution time. However, by using a hybrid approach, the amount of time needed to bring up an application can be reduced to reasonable levels. To do this, create a general-purpose dump file that captures a Lisp environment containing most of the code that your applications need to run. Then the script that runs a particular application does not need to load all of the support binaries.

Another drawback to using the script method described here is that it works only for application programs that do not read from HP-UX standard input. If your application does read from standard input, you should use one of the previously described methods, or have your users use a two step process to invoke your application:

1. Restore the general-purpose dump file.
2. Execute a Lisp form to load a file you've created that loads your application and calls the top-level function (and optionally calls `sys:exit`).

The steps to create a general-purpose dump file for use with HP-UX scripts are:

1. Execute the HP-UX command

```
lisp-loader -t 6500000 -b .55 -m 200000
```

2. Type in the following forms:

```
(in-package 'user)
(require "objects")
(require "hp-ux_2")
(require "hp-ux_3g")
(require "hp-ux_3w")
(require "windowutil" "$LISP/doc/examples/windowutil.b")
(system:save-world "$LISP/bin/appl-dump")
(system:exit)
```

Example Script

Once you have created the general-purpose dump file, you can write a script for each of the applications you want available. Let's assume again that we want to run the tic-tac-toe program. Here is a shell script that would achieve that purpose. The script assumes that the directory containing appl-dump (created above) is in your PATH shell variable.

```
: Sh, not csh
# Script to run Lisp tic-tac-toe game
appl-dump << EOF
; Lisp forms that you want to execute
(load "$LISP/doc/examples/tictactoe")
(ttt:tic-tac-toe)
(system:exit)
EOF
```

Put this text in a file (say \$HOME/bin/ttt) using an editor, and then make it executable with the HP-UX command

```
chmod +x $HOME/bin/ttt
```

Running the script (by typing its name) will run the tic-tac-toe game.

If you don't want anything to appear in the window from which you invoke the script, you can redirect the output to /dev/null by changing the third line to

```
appl-dump << EOF > /dev/null
```

MANUAL COMMENT CARD

LISP Application Notes
for HP 9000 Series 300 Computers
HP Part No. 98678-90010

Name: _____

Company: _____

Address: _____

Phone No: _____

Please note the latest printing date from the Printing History (page ii) of this manual and any applicable update(s); so we know which material you are commenting on _____.

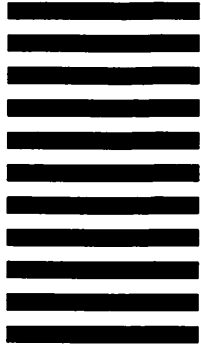


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 37 LOVELAND, COLORADO

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525







**HP Part Number
98678-90010**

Microfiche No. 98678-99010
Printed in U.S.A. 5/86



98678 - 90600
For Internal Use Only