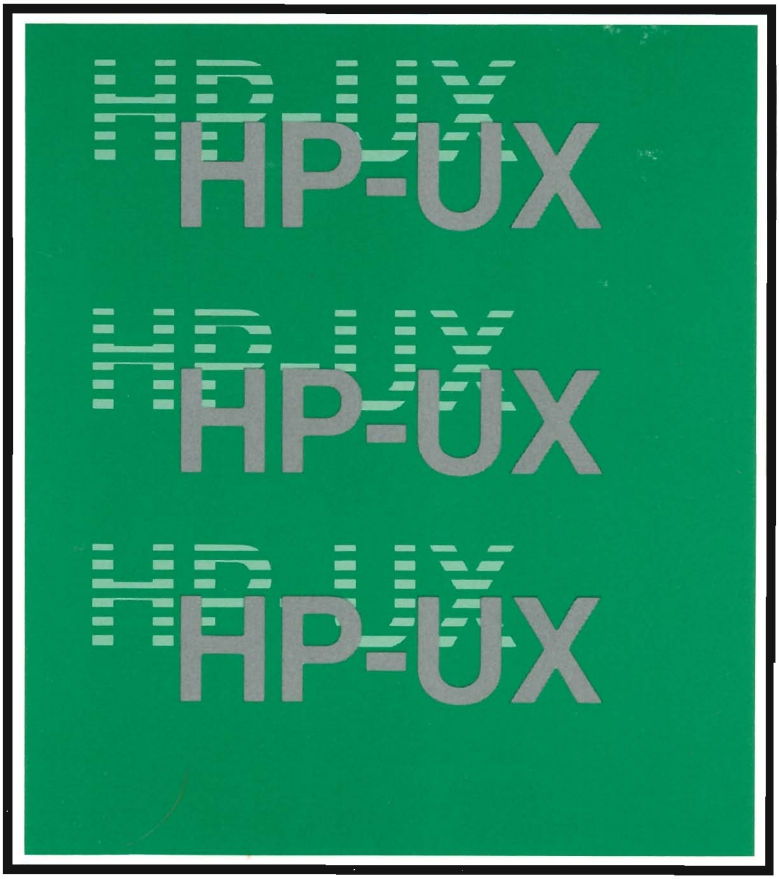
HP 9000 Series 200/500 Computers

HEWLETT PACKARD

HP-UX Portability Guide



HP-UX Portability Guide

for the HP 9000 Series 200/500 Computers

Manual Part No. 98680-90045

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.

October 1984...Edition 1

Table of Contents

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Chapter 4: Transporting Files

General Portability Guidelines

Chapter 1

Introduction

portable : capable of being carried or moved about.

Portability is a desirable software quality. When you develop a program, you want it to perform correctly in different environments without major changes. This book presents guidelines and techniques for maximizing the portability of programs written on and for HP 9000 computers running the HP-UX Operating System. We will discuss portability of high level source code (C, Pascal, FORTRAN).

Note

This edition of this manual applies to HP-UX Version 4.0 for the Series 500, and Version 2.1 for the Series 200.

The portability information is presented on two levels: software compatibility with non-HP systems, and compatibility between Hewlett-Packard's Series 200 and Series 500 computers. Porting to another manufacturer's machine involves unknowns and is only covered in general terms.

This manual also covers transporting data and source files between commonly used formats. It is assumed that you have some programming experience and are familiar with HP-UX.

This manual is divided into four chapters:

1. **Introduction**: Gives a brief description of this manual along with some general portability guidelines.
2. **Language Portability**: Contains sections for C, Pascal, and FORTRAN that describe areas of concern for portable programming.
3. **System Calls and Subroutines**: Describes variations in availability and behavior of system calls and subroutines between Series 200 and Series 500 HP-UX.
4. **Transporting Files**: Describes methods for moving source or data files between computer systems.

The Portable Philosophy

It takes the right attitude to develop portable software. Throughout the development process, be aware of things that could hinder porting your program to another environment:

- Non-standard language extensions.
- Assembly code.
- Hardware dependencies.
- Absolute addressing.
- Floating point comparisons.
- Software "tricks" that exploit a particular architecture.

The Merits of C

For ease of porting, we strongly recommend that you program in the C programming language. There are many advantages to using C on HP-UX and other UNIX[1] systems:

- Most of HP-UX is written in C. The system calls are implemented as C procedures. Consequently, C code can interface with HP-UX more easily than code written in other languages.
- C was designed to be portable. Machine dependence is minimized by isolating dependencies in library routines.
- Low-level operations such as bit manipulation are supported in a portable way. This reduces the need for assembly routines, but the procedures may still have to be changed when porting because the meanings of bit positions vary from machine to machine.
- Most implementations of C are "plain vanilla." There is no need to worry about using a feature that might not be available on another system.

1 UNIX is a trademark of AT&T Bell Laboratories.

Guidelines

Here are some guidelines for making your code portable:

- Isolate all machine dependent code in libraries. Maintain one for each execution environment, or use the conditional compilation features (`#ifdef`) of C.
- Read the section of this manual that describes the anomalies of the language you are using; these appear in Chapter 2. When writing your code, keep these variations in mind as potential problems. These sections also discuss adherence to language standards.
- Read the sections on system calls and subroutines in Chapter 3 and note the differences between Series 200 and Series 500. Some variations from other UNIX systems may be documented here, but the only way to be certain is by comparing the entries in your *HP-UX Reference* with your UNIX documentation.
- Don't take the easy way out. It may be simpler initially to use a language extension or hardware quirk to achieve your programming goals, but if you want your code to be portable, avoid shortcuts.

Language Portability

Chapter 2

Introduction

Since programming languages define the meaning of a program, they are the primary concern of portability. Unless the semantics of a language are exactly the same on two different machines, one cannot assume that a program written in that language will produce the same results on both machines. Also, an implementation of a language may support extensions that are not available on other systems. This chapter discusses areas you should be concerned with when porting programs in three of the languages available on HP-UX (C, Pascal, and FORTRAN).

Topics addressed are:

- Variations from language standards.
- Differences in HP-UX command line options.
- Variable storage.
- Calls to other languages.

Things to be Aware Of

In addition to semantic differences, you should be alert to variations in the way your system processes source code. This includes compiler directives and command line options. If you are using the *make* utility, you will probably have to alter the compiler options in your makefiles to reflect system differences.

Compiler directives are a mixed blessing. There are directives available on HP-UX that generate warnings for non-standard language features. These are very useful and are covered under the each language. On the other hand, there are directives that enable machine dependent features that dissolve any hope of portability. In any case, the directives will have to be changed when porting because it is unlikely that the systems you are porting between support the same directives. You must balance the current usefulness of the directive against its potential for portability problems.

Floating point operations are another fly in the ointment of compatibility. Computer floating-point numbers are usually only close approximations of real numbers, so when doing floating point compares, it is best to compare to a range of values instead of a single value. This technique is known as a "fuzzy compare." For example:

Replace

```
if x = 1.2267 then
     y:= y + 1;
```

with

```
if (abs(x - 1.2267) < err_margin) then
     y:= y + 1;
```

where `err_margin` is a constant representing the margin of error for comparisons.

What's Next

The rest of this chapter contains sections on C, Pascal, and FORTRAN that detail the portability aspects of each language.

The C Programming Language

C is the most portable programming language available under HP-UX. Carefully written C programs can be ported to other machines unchanged. This portability and its close ties to HP-UX make C the language of choice for programming in the HP-UX environment.

Additionally, HP-UX provides the *lint* utility, which detects type clashes and possible portability problems in your code. See *HP-UX Concepts and Tutorials* for details on using *lint*.

Another nice portability feature of C is `#include` files. Machine dependent code and declarations can be segregated in separate files, so that to port the code, you need only change some `#include` statements and supply the appropriate files to include.

C also has conditional compilation directives like `#ifdef` and `#ifndef` that can control compilation of machine dependent code sections.

C Dependencies

There are still some things that you need to be concerned with when writing portable programs in C. These include data sizes, parameter passing conventions, and the exact specification of some operations. In order to avoid subtle errors, you should be certain that the machine you are moving your programs to behaves the way that your programs expect. The following is a list of areas where the HP-UX implementation of C **may** deviate from other C compilers (including differences between Series 200 and 500). Each section is marked as to whether it applies to the Series 500, Series 200, or both.

Data Type Sizes (200 and 500)

This table shows the sizes of the six C data types:

Char	Size
char	8 bits
short	16 bits
int	32 bits
long	32 bits
float	32 bits
double	64 bits

The `typedef` facility is the easiest way to write a program to be used on machines with different data type sizes. Simply define your own type equivalent to a provided type that has the size you wish to use.

Example: Suppose Machine A implements `int` as 16 bits and `long` as 32 bits. Machine B implements `int` as 32 bits and `long` as 64 bits. You want to use 32 bit integers. Simply declare all your integers as type `MYINT`, and insert the appropriate `typedef`. This would be

```
typedef long MYINT;
```

in code for machine A, and

```
typedef int MYINT;
```

in code for machine B. `#include` files are useful for isolating the machine dependent code like these type definitions. For instance, if your type definitions were in a file `mytypes.h`, to account for all the data size differences when porting from machine A to machine B, you'd only have to change the contents of file `mytypes.h`.

Char Data Type (200 and 500)

The `char` data type defaults to signed. If a `char` is assigned to an `int`, sign extension takes place. A `char` may be declared `unsigned` to override this default. The line

```
unsigned char   ch;
```

declares one byte of unsigned storage named `ch`. On some systems, `char` variables are unsigned by default.

Register Data Type (200)

The `register` data type is supported on Series 200 HP-UX, and if properly used, will reduce execution time. Using this type should not hinder portability, however, its usefulness on other machines will vary, since some ignore it.

Register Data Type (500)

Because the Series 500 computers are stack machines, the `register` specification is harmlessly ignored.

Identifiers (200 and 500)

Identifiers can be as long as you want, but they have 255 **significant** characters. For universally portable code, use considerably less than this. Eight significant characters for internal identifiers and seven for external identifiers (identifiers that are defined in another source file) is safe. Although identifiers in HP-UX C are case sensitive, you should avoid identifiers that are unique by case only (ie. `Num1` vs. `num1`). Typical C programming practice is to name variables with all lower-case letters, and `#define` constants with all upper case.

Shift Operators (200 and 500)

On left shifts, vacated positions are filled with 0. On right shifts of signed operands, vacated positions are filled with 1 (arithmetic shift). Right shifts of unsigned operands fill vacated bit positions with 0 (logical shift). Integer constants are treated as signed unless cast to unsigned.

Bit Fields (200 and 500)

Bit fields are assigned left to right and are unsigned.

Division by Zero (200 and 500)

Division by zero gives the run-time error message `Floating exception.`

Integer Overflow (200 and 500)

As in nearly every other implementation of C, integer overflow does not generate an error. The overflowed number is "rolled over" into whatever bit pattern the operation happens to produce.

Null Pointers (200 and 500)

Some versions of C permit references to null pointers. In the HP-UX implementation of C, referencing a null pointer causes a run-time error. Since some programs written on other UNIX systems rely on being able to reference null pointers, you may have to change code to check for a null pointer. For instance, change

```
if (*ch_ptr != '\0')
```

to

```
if ((ch_ptr != NULL) && (*ch_ptr != '\0'))
```

Parameter Lists (200)
On the Series 200, parameter lists grow towards higher addresses; to use a pointer to step through a parameter list, increment the pointer. Example:

```
parprint (a,b,c)
        int a,b,c;
{
        int i, *ptr;

        ptr = &a;  /* SET POINTER TO ADD. OF FIRST PARAM */
        for (i = 1; i <= 3; i++)    /* PRINT EACH PARAM */
                {
                printf("\n %d",*ptr);
                ++ptr;
                }

}  /* END parprint */
```

Calling this function would print its three parameters in order.

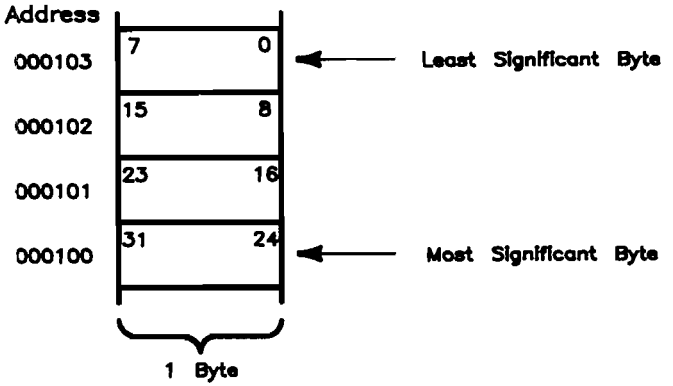
Parameter Lists (500)
On the Series 500, parameter lists are stacked towards decreasing addresses (though the stack itself grows towards higher addresses). To step through a parameter list, decrement the pointer. Example:

```
parprint (a,b,c)
        int     a,b,c;
{
        int     i, *ptr;

        ptr = &a;  /* SET POINTER TO ADD. OF FIRST PARAM */
        for (i = 1; i <= 3; i++)    /* PRINT EACH PARAM */
            {
            printf("\n %d",*ptr);
            --ptr;
            }

} /* END PARPRINT */
```

Calling this function will print its three parameters in order. Note that the only difference between this function and the similar one for Series 200 HP-UX is that `ptr` is decremented instead of incremented.

Memory Organization (200 and 500)

On both the Series 200 and Series 500 computers, the most significant byte of a datum has the lowest address. This is the address used to access the datum.



Expression Evaluation

The order of evaluation for some expressions will differ between Series 200 and Series 500 computers. This does not mean that operator precedence is different. For instance, in the expression `x1 = f(x) + g(x) * 5;`, `f` may be evaluated before or after `g`, but `g(x)` will always be multiplied by `5` before it is added to `f(x)`. It is good programming practice to disambiguate all expressions with parentheses. Since there is no C standard for order of evaluation of expressions, avoid using functions with side effects and function calls as actual parameters. Use temporary variables if your program relies upon a certain order of evaluation.

Parent and Self Directory Entries

On Series 500 HP-UX, directories do not contain entries for . and .. (current and parent directories). Any program that relies on those entries being present will not work.

Compiler Command Options

There are some minor differences between Series 200 and Series 500 C compiler options. If you are using *make*, you may have to change the compile lines in your makefiles when porting your code. Here is a list of the variant options. See the HP-UX Reference for more details.

Option	Effect	Difference
-a	Option to assembler	Series 200 only.
-b	Floating point hardware option	Series 200 only.
-B	Substitute compiler passes	Series 200 and 500, but no substitute compiler passes are provided for either.
-f	Floating point hardware option	Series 200 only.
-F	Enable program analysis	Series 500 only.
-g	Debug on	Currently Series 500 only.
-N	Adjust table sizes	Series 200 only.
-p	Enable profiling	Currently Series 200 only.
-t	Pass Option	Supported by both 200 and 500, but two additional pass designators (a, l) are available on Series 200.
-v	Verbose	Supported by both 200 and 500, but may not be available on other UNIX systems.

Series 200 Floating Point Options

If your programs will be run on Series 200 computers with optional floating point hardware, you must use one of two compiler options to take advantage of its faster processing.

- **−b** causes the compiler to generate code for floating point operations that will use the special hardware if it is present at run-time.
- **−f** causes the compiler to generate code to use the special hardware. The code will not run on a machine without the floating point hardware.

The **−b** option is the preferred form if you are not concerned with the extra space or time the code will take to decide whether there is a floating point card present. If you use **−f**, then the code can only be used on machines with the optional floating point hardware.

Calls to Other Languages

It is possible to call a **function** written in another language from a C program, but you should have a good reason for doing so. Using more than one language in a program that you plan to port to another machine will complicate the process. In any case, make sure that the program is thoroughly tested in any new environment.

If you do call another language from C, you will have the other language's anomalies to consider plus possible variances in parameter passing. Since all HP-UX system routines are C programs, not calling programs in other languages should not be a hardship. But if you choose to do so, remember that C passes all parameters by value. The ramifications of this depend on the language of the called function.

Pascal

Pascal gives you the choice of passing parameters by value or by reference (`var` parameters). C passes all parameters by value, but allows passing pointers to simulate pass by reference. If the Pascal function does not use `var` parameters, then you may pass values just as you would to a C function. Actual parameters in the call from the C program corresponding to formal `var` parameters in the definition of the Pascal function should be pointers.

Unfortunately, calling Pascal functions from C on the Series 200 is different than on the Series 500.

On the Series 200, you must put the Pascal function in a module that exports the function, compile that file, and then link it with your main C program by including the name of the Pascal .a file on the *cc* command line.

On the Series 500, use the $SUBPROGRAM$ directive in the file containing your Pascal function instead of putting the function in a module.

Example 1:

This example shows the code to call a Pascal function from a C program on the Series 200. An example for the Series 500 follows.

```
main ()
/*  CALL A PASCAL FUNCTION */
{
        int     a, b, psubs_changenum();

        a = -4;
        b = 3;

        printf ("\n Before the call, a = %d", a);
        psubs_changenum (&a, b); /* NOTE THE USE OF & FOR VAR PARAM */
        printf ("\n After the call, a = %d", a);

}
```

Source for Main C Program (`main.c`)

```
module psubs;

export
        function changenum (var num1: integer; num2 : integer): integer;

implement

function changenum (var num1: integer; num2 : integer): integer;
{ FUNCTION TO ADD NUM1 TO NUM2, IF THE NEW VALUE OF NUM1 IS
  NEGATIVE THEN 0 IS RETURNED, OTHERWISE 1 IS RETURNED.      }

begin
        num1:= num1 + num2;
        if num1 < 1 then
            changenum:= 0
        else
            changenum:= 1
end; { CHANGENUM }

end. { MODULE PSUBS }
```

Pascal Function Source (`psubs.p`)

The commands to compile these files into the executable file a.out are:

```
PC -C Psubs.P
CC main.c Psubs.a -lPC
```

The -lPC tells the C compiler to link any required object files from the Pascal library.

Note that the C program refers to the Pascal function as Psubs_changenum. This follows the general form of <module name>_<function name>.

If you want to access I/O functions from a Pascal function within a module, you must declare the files you wish to use (including input and output) in the function.

Example 2: This example shows the code to call a Pascal function from a C program on Series 500 HP-UX.

```
main ()
/*  CALL A PASCAL FUNCTION */
{
        int     a, b, changenum();

        a = -4;
        b = 3;

        Printf("\n Before the call, a = %d", a);
        changenum (&a, b); /* NOTE THE USE OF & FOR VAR PARAM */
        Printf("\n After the call, a = %d", a);

}
```

Source for Main C Program (main.c)

```
$subProgram$

Program dummy(input,output);

function changenum (var num1 : integer; num2 : integer): integer;
 { FUNCTION TO ADD NUM1 TO NUM2, IF THE NEW VALUE OF NUM1 IS
   NEGATIVE THEN 0 IS RETURNED, OTHERWISE 1 IS RETURNED.  }

begin
        num1:= num1 + num2;
        if num1 < 1 then
            changenum:= 0
        else
            changenum:= 1
end; { CHANGENUM }

. { END COMPILATION UNIT }
```

Pascal Function Source (Psubs.P)

The commands to compile these files into the executable file `a.out` are:

```
PC -c psubs.P
cc main.c psubs.o -lPC
```

The `-lPC` tells the C compiler to link any required object files from the Pascal library.

FORTRAN

No example is given here for calling a FORTRAN function from a C program. The section on calling C from FORTRAN may be helpful. Remember that in FORTRAN, all parameters are passed by reference, so actual parameters in a call from C must be pointers, or variable names preceded by the address-of operator (&).

You can compile FORTRAN functions separately by putting the functions you want into a file and compiling it with the **-c** option to produce a .o file. Then, include the name of this .o file on the command line that compiles your C program. The C program can refer to the FORTRAN functions by the names they are declared by in the FORTRAN source.

Pascal

Both the Series 500 and 200 HP-UX systems support a version of Pascal known as Hewlett-Packard Standard Pascal (HP Pascal). HP Pascal is a superset of ANSI Pascal, and implements many advanced features. A few of the features differ between the Series 200 and 500; the differences are covered in this section.

The extensions of HP Pascal are a blessing and a curse. If you plan only to run your programs on HP computers (better yet, only HP 9000 computers), then it won't take much work to move them, and the extra features will make your programming much easier. **However**, if you should decide to port those programs to another manufacturer's computer, the effort to do so will be proportional to the use of non-standard Pascal extensions. Even if the system you are moving the programs to has extensions, it is doubtful that they have the same form as HP Pascal. Before deciding to use a non-ANSI feature, ask yourself some questions:

- Am I **ever** going to port this program to a non-HP machine?
- How much hardship does avoiding the extension cause?
- Will another machine have a similar feature?

If your answers are "probably not," "a heck of a lot," and "I sure hope so," then go ahead and use the extension.

How can you know whether any of the language features you are using are likely to be supported on another machine? Both Series 200 and Series 500 Pascal have an option that causes the compiler to emit warnings for uses of features not included in ANSI Standard Pascal. On either machine, include the line

```
$ANSI ON$
```

at the beginning of your source file. You will have to use the **-L** option with *pc* and look at a listing of your program (on the screen or hardcopy) to see where the warnings occurred.

Series 200 HP-UX vs. Series 500 HP-UX

The pc Command

There are some minor differences in the Pascal compiler (*pc*) between Series 200 and Series 500 HP-UX.

The Series 500 has a few options not currently available on Series 200:

-W	Display or set a program's working set size.
-H	Display or set a program's maximum heap size.
-E	Link with /lib/libpcesc.a.
-g	Produce debug information.
-F	Produce information for program analysis.

The Series 200 Pascal compiler produces .a files, while the Series 500 Pascal compiler produces .o files. For instance:

```
PC main.P util.P
```

would produce the files `main.a`, `util.a`, and `a.out` on Series 200 HP-UX and the files `main.o`, `util.o`, and `a.out` on the Series 500. This means that any *make* files or shell scripts that depend on the code file suffix will have to be changed to port them between the two series.

Compiler Option Differences

Series 200 and Series 500 HP-UX Pascal support different (although intersecting) sets of compiler options. Additionally, some common options have different semantics, and a slightly different syntax. For portable code, keep compiler options to a minimum. Especially avoid ones that affect the semantics of the language or enable system level programming extensions, like `$PARTIAL_EVAL$` and `$SYSPROG$` on the Series 200.

On the Series 500, more than one option can appear between a pair of dollar signs. The 200 allows only one.

Here is a list of the variant options:

AUTOPAGE	Series 500 only. Controls pagination of listing.
CODE	Series 200 only. Selects whether a code file is generated.
DEBUG	Available on both machines, but prepares more information on Series 500.
FLOAT_HDW	Series 200 only. Controls generation of code for floating point hardware.
IDSIZE	Series 500 only. Specifies number of significant characters in identifiers.
IF	Series 200 only. Conditional compilation.
LINE_INFO	Series 500 only. Listing option.
LINENUM	Series 200 only. Specifies source line number.
LINESIZE	Series 500 only. Controls line buffering for TEXT files.
LIST	On the Series 200, you can specify where the the listing is to be directed.
OVFLCHECK	Series 200 only. Switches overflow checking on or off.
PAGEWIDTH	Series 200 only. Controls width of source listing. See WIDTH for Series 500.
SAVECONST	Series 200 only. Controls scope of structured constants.
SEARCH_SIZE	Series 200 only. Changes number of external files that can be searched.
SKIP_TEXT	Series 500 only. Skips source text.
STANDARD_LEVEL	Series 500 only. Sets level of extensions that can be used without triggering a warning. Series 200 uses ANSI and SYSPROG.
STATS	Series 500 only. Display compiler options.
SUBPROGRAM	Series 500 only. Separate compilation facility.
SUBTITLE	Series 500 only. Prints a listing subtitle.

SYSPROG	Series 200 only. Allows use of system programming extensions. Equivalent to $STANDARD_LEVEL 'hp_MODCAL'$ on the Series 500.
TITLE	Series 500 only. Prints a listing title.
TYPE_COERCION	Series 500 only. Relaxes type checking.
UPSHIFT_LEVEL1	Series 500 only. Makes all external names uppercase.
VISIBLE	Series 500 only. Specifies default entry points.
WIDTH	Series 500 only. Equivalent to PAGEWIDTH on Series 200.

Differences in Features

Due to the varying origins of the Series 200 and Series 500 Pascal compilers, there are some differences between them. Fortunately, most of the differences are in what features are available, so that the same Pascal source code can be used on both machines as long as you avoid the features that are only available on one of the machines. Here is a list of the features that differ between Series 200 and Series 500 HP-UX Pascal.

Conformant Arrays	Series 500 only.
Type Coercion	Series 500 only.
Absolute Addressing	Series 200 only.
Enumerated Type I/O	Series 200 only.
Sizeof	Series 200 Pascal allows using a file variable as a parameter to the `sizeof` function; Series 500 does not.
Try-Recover	Escape codes for errors differ between the two Series.
Function Types	Series 500 Pascal has both procedure and function types, while Series 200 has only procedure types. Assignments to procedure variables have a different syntax.
String Length	Maximum string length in Series 500 Pascal is 32 767 characters; on the Series 200, it is 255 characters.
Packed Arrays	On the Series 200, elements of packed arrays cannot be passed as `var` parameters.

Series 200 HP-UX vs. Series 200 Workstation

Since the Series 200 HP-UX Pascal compiler was developed from the HP Pascal workstation, the two implementations are very similar. There are still some differences that you should be aware of when porting between the two systems. If your programs to be ported use operating system dependent features like low-level I/O functions, then you may have a non-trivial porting job.

The information in this section is covered in greater detail (though with different organization) in the *HP Pascal Language Reference* for Series 200 computers.

Compiler Option Differences

The options available on HP-UX Series 200 Pascal are a subset of the ones available on the Pascal workstation implementation. The following options are available **only** on the Pascal workstation.

CALLABS	Switches absolute jumps on and off.
COPYRIGHT	Includes copyright information.
DEF	Changes size and location of compiler's .DEF file.
HEAP_DISPOSE	Controls garbage collection.
IOCHECK	Controls error checking on system I/O routine calls.
REF	Changes size and location of compiler's .REF file.
STACKCHECK	Controls stack overflow checking.
SWITCH_STRPOS	Switches order of parameters for the STRPOS function.
UCSD	Allows use of UCSD Pascal extensions.

Differences in Features

There are some minor semantic differences between the workstation and HP-UX Pascal implementations.

Module Names	Module names on HP-UX can be up to 12 characters, while on the Pascal workstation they can be up to 15.

Real Variables	Real variables are 32 bits in HP-UX Pascal and 64 bits on the workstation. Longreals are 64 bits on both implementations.
Input	Although HP Standard Pascal specifies unbuffered input, on the HP-UX implementation, input is buffered by default. To override this, add the following statement to the beginning of your program: `reset(input,'','unbuffered');`
Lastpos	Not implemented on Series 200 Pascal workstation.
Linepos	Not implemented on Series 200 Pascal workstation.
Heap Management	The Series 200 HP-UX and Pascal workstation have different mechanisms for specifying the heap manager. See the *HP Pascal Language Reference* for the details of using them.
Packed Arrays	In Series 200 HP-UX Pascal, elements of packed arrays cannot be passed as `var` parameters.

Calling C Functions From Pascal

HP-UX system calls and subroutines are defined as C functions, so you may need to call a C function from a Pascal program. Fortunately, Pascal and HP-UX are flexible enough to make this a simple operation. This section contains a list of concerns and some examples of calling a C function from a Pascal program.

C Functions

All subprograms in C are functions that return a result. The default type of the returned value is integer, but real values and pointers may also be returned. Since the C function will not be defined in the same source file as your Pascal program, you will have to declare the C function as an external Pascal function within the source file. It is important for you to make the external declaration correspond to the definition of the C function.

Parameter Passing

Pascal gives you the choice of passing parameters by value or by reference. C passes all parameters by value, but can emulate pass by reference by declaring a formal parameter to be a pointer. This relationship is important to understand when writing the external function declaration through which Pascal "sees" the C function. If the C function you are calling has a formal parameter declared as a pointer, then in your Pascal external declaration of the function, the formal parameter should be a `var` parameter. All C formal parameters that are not declared as pointers should have corresponding Pascal non-`var` parameters. See the example below for clarification.

Data Compatibility

This chart shows equivalent Pascal data types for given C types:

C Type	Equivalent Pascal Type
int	integer
char	char
float	real
double	longreal
character array	packed array of char

Records and structures can be easily passed between C and Pascal as long as the Pascal records are unpacked. Packed records introduce problems that are not discussed here. Both C and Pascal store arrays in row-major order so they may be passed. When passing character arrays (which are actually pointers to chars), make sure that they are terminated with chr(0). Always be sure to debug the interface between the two languages. Don't assume that it works just because the function works when called by a program in the same language.

Alias

If you want to refer to an external function by a name other than the one it is defined under, use the `alias` directive. This technique is shown in Example 2.

Example 1

This example illustrates calling a user defined C function from a Pascal program.

```
{ SHORT PROGRAM TO CALL C FUNCTION }
program call_c(input,output);

const   str_length = 50;

type mystring = packed array[1..str_length] of char;

var     x : real;
        s : mystring;

{ DECLARE THE C FUNCTION AS AN
  EXTERNAL PASCAL FUNCTION }
function c_sub (var strng : mystring): real; external;

begin
   s:= 'abc';
   s[4]:= chr(0); { PUT NULL AT END }
   x:= c_sub(s);  { CALL THE FUNCTION }
   writeln(x)
end.
```

Pascal Source (`call_c.p`)

```
#include <stdio.h>
/* C FUNCTION TO PRINT A STRING
   AND RETURN A REAL VALUE. */
float c_sub(str)
        char    *str;
{
   printf("\n %s",str);
   return(1.211);
}
```

C Source (`c_sub.c`)

The procedure for compiling and linking these two source files is:

```
cc -c c_sub.c
pc call_c.p c_sub.o
```

Then executing the file `a.out` would produce:

```
abc         1.211000E+00
```

Example 2

This example calls the HP-UX system function *truncate* from a Pascal program. The `alias` directive is used to rename the external symbol `truncate` to `chop` within the program. Note the section that inserts a null (chr(0)) into the character array at the end of the file name. This is necessary because C expects all strings to be terminated by a null.

```
program chopfile(input,output);
{ PROGRAM TO TRUNCATE A FILE TO A GIVEN LENGTH }

const   str_length = 50;

type    mystring=packed array[1..str_length] of char;

var     fname : mystring;
        lngth, dummy, i : integer;

function $alias 'truncate'$ chop(var path : mystring;
                  length : integer); integer; external;

begin
   writeln('Enter name of file to be chopped: ');
   readln(fname);

   { PUT NULL IN FIRST SPACE }
   i:= 1;
   while (fname[i] <> ' ') do
      i:= i + 1;
   fname[i]:= chr(0);

   writeln('Enter new length: ');
   readln(lngth);

   { CALL THE SYSTEM FUNCTION
      WITH ITS ALIASED NAME }
   dummy:= chop(fname,lngth);

   if dummy <> 0 then
      writeln('CALL FAILED')

end. { CHOPFILE }
```

Use these commands to compile and run this program:

```
pc chopfile.p
a.out
```

FORTRAN

If you will be porting many FORTRAN programs written under HP-UX then you should get a copy of the HP publication *FORTRAN/9000 Comparison Notes*. It extensively documents the differences between Series 200, Series 500 and ANSI FORTRAN. This section of this manual covers major environmental differences between the two HP-UX FORTRAN implementations, such as compiler directives and command line options. The major language "gotchas" are documented here, though not in fine detail. This section also describes (with examples) how to call a C function from FORTRAN.

Compile Command

The Series 200 and Series 500 each use a different command to compile FORTRAN programs. The Series 200 uses the UNIX standard **f77**, while the 500 uses **fc**. If you're working on a Series 500 HP-UX system, we recommend that your system administrator link **f77** with **fc** to standardize the command. Series 500 HP-UX permits options to appear anywhere on the command line after **fc**, but Series 200 requires that the options precede the names of the source files.

Compiler Options

The Series 200 and 500 support different compiler command line options. Here is a list of the options that vary between the two systems. See the HP-UX Reference for more details.

Option	Effect	Difference
-b	Floating point option	Series 200 only
-D	Compile debug lines	Series 500 only
-e	Write errors to stderr	Series 500 only
-f	Floating point option	Series 200 only
-F	Enable program analysis	Series 500 only
-g	Debug on	Series 500 only
-I4	Data size option	Series 200 only
-k	Dynamic local arrays	Series 200 only
-L	Listing to stdout	Series 500 only
-N	Adjust table sizes	Series 200 only
- onetrip	Alter DO semantics	Does not affect implied DOs on Series 200
-p	Prepare for profiling	Series 200 only
-Q	Specify option file	Series 200 only
-S	Compile; don't assemble	Series 200 only
-u	Implicit typing off	Can be overridden in a program unit on Series 200
-U	Case is significant	Handled differently by 200 and 500, not recommended
-Vc	Virtual COMMONs	Series 500 only
-Vd	Virtual SAVEs and DATA	Series 500 only
-Vf	Virtual FORMATs	Series 500 only
-w66	Suppress FORTRAN 66 warnings	Series 200 only

Compiler Directives

This section points out some FORTRAN compiler directives that could cause some portability problems. No attempt is made to list which directives are implemented in Series 200 and Series 500. For a complete discussion, see *FORTRAN/9000 Comparison Notes*, or consult your language references to see what directives are available on each system.

Since compiler directives are highly implementation dependent, using as few directives as possible will increase the portability of your code. Directives can be isolated in separate files, but accessing the files from your source requires using a directive. This is still the recommended strategy for handling directives, since (hopefully) most systems you port to will provide some mechanism for including files.

Here are some directives to beware of:

INCLUDE: This works similarly on both computers, except that each uses a different search path. On Series 500 machines, the search order is:

1. The current source directory
2. The current working directory
3. /usr/include

On the Series 200, the search order is:

1. The current working directory
2. /usr/include

Specify an absolute path name (starting with /) in the INCLUDE line to achieve search path independence.

ALIAS: The meaning of the ALIAS directive differs slightly between the Series 200 and Series 500. On Series 500, ALIAS allows you to specify the parameter passing mechanism to be used when calling the aliased procedure. This is not supported on Series 200; instead, use the "onionskin" technique for foreign language calls described later in this chapter.

Semantic Differences and Extensions

The HP implementations of FORTRAN are not of the plain vanilla variety. Extensions have been made to the language that may not be available on other systems. These extensions are convenient if you plan to run your programs exclusively on one system, but they can be a real headache to port. When programming, you must balance the current utility of the extension against its potential for portability problems.

Use of extensions is not a concern if you plan only on porting between Series 200 and 500. The differences between the two are minor and are fully documented in *FORTRAN/9000 Comparison Notes*. Some important things to be aware of are:

- The Series 200 and 500 use different algorithms for real arithmetic.
- Use end of line comments only in columns 1-72 on non-continued lines.
- Series 200 string constants may not exceed 255 characters. The Series 500 supports strings of any length.
- Names have 255 significant characters on both machines, but Series 500 names can be any length.
- Arrays in Series 200 FORTRAN can have at most 20 dimensions. Series 500 arrays are limited only by the amount of storage available.
- Statement ordering is more strictly enforced on the Series 500. (See Figure 3-1 of either FORTRAN 9000 reference manual.)

If you will be porting to a non-HP system, then avoid using language extensions. Inserting the line

```
$OPTION ANSI ON
```

at the beginning of your source will make the compiler include in the listing warnings for uses of features that are not a part of the ANSI 77 standard.

The HP 9000 implementations of FORTRAN support the Military Standard Definition (MIL-STD-1753) of extensions to the ANSI 77 Standard. See your language references for details on what extensions the standard includes.

Recursion ?!

One major feature of HP's versions of FORTRAN is that they support recursion. This means that variable storage for subroutines and functions is dynamic (except for local arrays on Series 200 systems). Hence, variables in subprograms do not retain their values between invocations. If you are writing code on a Series 200 or 500 to use on another system, do not use recursion. If you are moving FORTRAN code from another system to an HP machine, use the SAVE statement in all subprograms, or compile with the **-K** command line option to achieve the same effect.

Since local arrays are handled differently between Series 200 and Series 500, you must beware when porting recursive programs between them. Either don't use arrays in recursive subprograms or use the **-k** option when compiling on the Series 200 to force dynamic allocation of local arrays. Remember that there is a 32K byte limit on local dynamic storage on the Series 200.

Calling C Routines From FORTRAN

Since all the HP-UX system calls and subroutines are accessed as C functions, you may want to call a C function from a FORTRAN program. There are some basic obstacles to doing so. The major problem is that C and FORTRAN pass parameters differently – C by value and FORTRAN by reference. If you are programming on a Series 500 machine, you can use the ALIAS directive to change FORTRAN's passing mechanism. However, in the following example, we cover the more general (and portable) "onionskin" technique. The example shows code to call one of the HP-UX bessel functions from a simple FORTRAN program.

Example:

```
$option ansi on
program callc

integer n
real*8  x, my_jn

n = 4
x = 4.0
print*, my_jn(n,x)  ; CALL C ONIONSKIN FUNCTION

stop
end
```

FORTRAN source to call a C function

```
#include <math.h>

double my_jn (my_n, my_x)
          int    *my_n;
          double *my_x;
{
        return( jn (*my_n, *my_x));
}
```

C "onionskin" function

The C function `my_jn` merely converts FORTRAN's call by reference (pointers) into a call by value to the desired C routine. Note that if you were writing your own C routine to call from FORTRAN, you could have declared the parameters as pointers and would not need an intermediate function. Similarly, if a formal parameter in the called C routine is declared a pointer, it can be passed through the intermediate routine unchanged.

For the example above, assuming the code is in files `callc.f` and `my_jn.c` respectively, the commands to compile and load the FORTRAN program are:

```
cc -c my_jn.c
f77 callc.f my_jn.o -lm
```

The resulting object file would be left in `a.out`. For the Series 500 substitute `fc` for `f77`.

Here is some helpful information for calling C functions from FORTRAN:

Logicals	C uses integers for logical types. A FORTRAN 2-byte LOGICAL is equivalent to a C short integer, and a 4-byte LOGICAL by a long or regular integer. In both C and FORTRAN, zero is false and any non-zero value is true.
Files	File units and pointers cannot be passed between C and FORTRAN. However, a file created by a program written in either language can be used by a program of the other language if the file is declared and opened in the latter program.

Character Data

Passing character data from FORTRAN to C is tricky because these languages represent character strings in completely different ways. The trick to doing it is to "equivalence" the character variable to a one-dimensional integer array and then pass the array to the C function. Since FORTRAN integers are 4 bytes long, the number of elements in the integer array should be the ceiling of the length of the string divided by four. FORTRAN passes the array by reference, so the corresponding parameter in the C function should be declared a **pointer** to a character.

This technique (illustrated below) works on both Series 200 and 500 HP-UX. However, some FORTRAN 77 compilers may not allow you to equivalence character variables to integers.

Example: This example shows passing a character string from a FORTRAN program to a C function. The function returns the number of characters in the string before a space or null.

```
      character       strng*35
      integer         tempch(9), chcount
      external        chcount
      equivalence     (strng, tempch)

      print*, 'ENTER STRING: '
      read 100, strng
100   format (a35)

      strng = strng // char(0)
      print*, 'THE STRING IS ', chcount(tempch), ' LONG'

      end
```

Main FORTRAN Program (`main.f`)

```
chcount (str)
        char    *str;
{
        int i = 0;

        while ((str[i] != ' ') && (str[i++] != '\0')) ;

        return (i);
}
```

C Function (`chcount.c`)

The commands to compile and link these two files are:

```
cc -c chcount.c
f77 main.f chcount.o
```

The resulting object file would be left in `a.out`. On the Series 500, substitute `fc` for `f77`.

System Calls and Subroutines

Chapter 3

This chapter documents differences in system calls and subroutines between Series 200 and Series 500 HP-UX. If you are porting from another UNIX system, be aware that HP-UX may not support the same set of system calls and subroutines. Absolutely no attempt is made here to document semantic differences between HP-UX and UNIX routines of the same name, but it is unlikely that there are any substantial differences.

The first part of this chapter lists in alphabetical order all the system calls that have differences. The second part covers subroutines. If you need more detail than what is given here, look up the routine in question in the *HP-UX Reference*. If there are any hardware dependencies for a routine, they will be listed.

Note

The information in this chapter was accurate at the time this book was printed. Updates and improvements to the HP-UX system may invalidate some entries in this section. The *HP-UX Reference* is the final word on what routines are available on a particular system.

System Calls

brk	Due to architectural differences, this has Series 500 hardware dependencies. See the *HP-UX Reference* under *brk*(2).
ems	Series 500 only.
errinfo	Series 500 only.
errno	Two additional *errno* values are implemented on the Series 500.
exec	The system imposed limit on the size of *argv* and *envp* is much larger than 5120 bytes on both HP-UX implementations. The System 500 and System 200 have different object module formats that may affect use of *exec*.
exit	On both Series 200 and 500, accounting is not currently supported, and the file **sys/proc.h** is not provided.
fork	*Fork* will fail on the Series 200 if there is not enough swapping memory to create the new process (ENOSPC). On the Series 500 it will fail if there is not enough physical memory to create the new process (ENOMEM).
ioctl	The include file **sys/ioctl.h** is not currently supported on the Series 500. The two series support different sets of ioctls. For instance, Series 200 HP-UX has ioctls for kernel profiling and tracing that are not available on Series 500.
link	On the Series 500, for Structured Directory Format (SDF) discs, if *path2* is "..", then that directory's i-node will be altered such that its ".." entry points to the directory specified by *path1*.
memadvise, memallc, memfree, memchmd, memlck, memulck, memvary	Series 500 only.

mknod

On the Series 500 there is an additional value – 0110000 – available under file type that specifies network special files. HP-UX also allows the value 0150000 to specify SRM type files.

nice

Some HP-UX process priorities are mapped into the same internal process priority resulting in reduced priority granularity.

open

The following items pertain to the Series 500:

- Execute and write access are mutually exclusive.
- Shared program files remain open for execution as long as there is a process executing the program.
- Once a shared program file with its sticky bit set has been loaded, it appears to be open indefinitely, even if the number of processes executing the program drops to zero.
- Demand loaded program files that are not shared remain open until all of the code and data have been loaded.

profil

Series 200 only.

ptrace

Series 200 only.

read

The following items pertain to the Series 500:

- A value of *nbyte* greater than 524 288 is not supported when *fildes* is associated with a device file unless the device is a terminal or the null device, or *buf* points to a local buffer and has been locked with *memlock*(2).
- Any request for greater than 0.5 megabytes on unsupported device files results in *errno* being set to EINVAL. Requests for less than 0.5 megabytes could result in *errno* being set to ENOMEM.

signal	There are extensive implementation differences between the Series 200 and 500. See the *HP-UX Reference* for details. Additionally, on the Series 500, odd values of signal addresses that are greater than one are assumed to be valid signal handler addresses.
stat	In the case of special files which refer to discs, **st_size** either returns the total physical size (in bytes) of the mass storage volume when appropriate, or -1 otherwise. This is a property of the physical device, not any directory structure imposed upon it.
times	For Series 500 computers with multiple CPUs, the child CPU times listed can be greater than the actual elapsed real time, since the CPU time is counted on a per-CPU basis.
trapno	Series 500 only.
uname	On HP-UX the first character of the *version* field is set to "A" for single user, "B" for multi-user (16 users), and "C" for multi-user (32 users).
ustat	On the Series 500, **f_fname[6]** is the driver name, not the file system name.
vfork	Series 500 only, though on Series 200 systems, *vfork* is aliased to *fork*.
vsadv	Series 500 only.
vson	Series 500 only.
write	The Series 500 has some anomalies that are listed in the *HP-UX Reference*.

Subroutines

abs	On HP-UX, calling *abs* with the most negative number returns that number.
directory	Series 200 uses *malloc* to allocate memory, while the Series 500 uses *memallc*.
end	The following items pertain to Series 500 HP-UX: • *End* is the lowest heap address available to the user; *etext* is the lowest available address in the D-data segment; *edata* is the first available address in the I-data area. • Use *memallc* instead of *malloc* to set the program break. • In C these names must look like addresses (ie. `&end` instead of `end` to access the value of *end*).
gpio_get_status	Currently Series 500 only.
gpio_set_ctl	Currently Series 500 only.
hpib_***	The *hpib* subroutines are currently available only on the Series 500.
intrapoff	Series 500 only.
io_***	The *io* subroutines are currently available only on the Series 500.
monitor	Series 200 only.
nlist	Series 200 only.
perror	The Series 500 provides the additional error indicator *errinfo*.
setjmp	On the Series 500, variables declared as `register` remain defined upon returning from a *setjmp* call caused by a *longjmp*. Note that depending on these variables is discouraged since different machines may treat register variables differently.
ssignal	Series 200 only.

string

On the Series 200 the argument *n* is limited to the amount of physical memory; on the 500, it is limited to about 500 Mbytes.

trig

The approximate limit for the values returned by these functions is 1.49×10^8.

Transporting Files

Chapter 4

Introduction

Portable source code isn't much good unless there exists a means of moving it to a different computer system. Yes, you could just get a listing and then type it in again on the new computer, but computers should automate such mundane tasks. Unfortunately, there are more ways of storing files than there are operating systems.

This chapter describes some methods and HP-UX commands for transporting data files to or from an HP-UX system. There is no way to cover every possible file transporting situation, but it explains the ones you are likely to encounter. The first four sections of this chapter discuss moving files between systems not connected by a data communications link. Moving files between computers via modems, phone lines, and datacomm links is covered in the last section.

For the sake of this discussion, computer systems can be divided into four categories:

- HP-UX systems
- UNIX or UNIX-like systems
- Other HP systems
- Others (Any system not in one of the above classes)

The table below shows HP-UX commands appropriate for moving files to/from each of the four categories.

	HP-UX	Other HP Systems	UNIX-like Systems	Others
tcio/cpio	x	-	-	-
cpio	x	-	x	-
tar	x	-	x	-
uucp	x	-	x	-
lifcp	x	x	-	-
srmcp	x	x	-	-
dd	x	x	x	x

Moving Between HP-UX Systems

Moving files between HP-UX systems is straightforward whether you're moving between two systems of the same series or not. It is assumed that the source and destination files are not on the same file system. If they are, use the *cp* command to copy the file. If the source and destination are not on the same file system, and there is not a **datacomm** connection between them, you will have to use an intermediate medium (floppy disc, 9-track tape, CS/80 cartridge).

If you're using an intermediate medium, transporting will be a two-stage process:

1. Source → Temporary medium
2. Temporary medium → Destination

The same command is used for both stages. The command you use depends on your intermediate medium.

Using Tar

The *tar* command can be used to transfer HP-UX files to or from a raw storage medium such as tape or initialized floppy disc. This example uses the device file for a floppy disc drive, but is easily adapted to other media by naming a different device file in the command line. The **key** (`cvf` on the first command line shown below) can also be changed to reflect your particular situation. See the *HP-UX Reference* for details.

1. To move the files `zonk.c` and `dynamo.c` from one HP-UX system to another, put an initialized floppy disc in the drive corresponding to the device file you will name in the command line and type:

   ```
   tar cvf /dev/rfd0 zonk.c dynamo.c
   ```

   Since `v` (verbose) was specified in the key, *tar* will echo the names of the files as they are written to the disc.

2. After the HP-UX prompt appears, remove the disc and insert it into the destination system's disc drive. On a terminal connected to this system type:

   ```
   tar xf /dev/rfd0 zonk.c dynamo.c
   ```

   The files will be transferred to the current working directory.

A similar two stage process could be performed with an initialized floppy by using *lifcp*. This command is discussed under *Trading Files with Other HP Systems*.

Trading Files with Other UNIX Systems

Files can be exchanged between HP-UX and other UNIX systems in much the same way as between two HP-UX systems. The example above using *tar* could be altered to use tape drives to move files to or from another UNIX system. Another alternative is *cpio*.

Using Cpio and Tcio

Cpio works similarly to *tar*. The major differences are that *cpio* gets the names of files to be copied from standard input instead of the command line, knows about special files, and does not automatically recurse through directories. Like *tar*, what intermediate medium you use to transfer the files depends on what storage devices are supported on your systems. However, if you're using a CS/80 data cartridge, use *tcio* with *cpio* to save wear on the tape and drive. Here is an example that uses a 9-track tape to transport some files to an HP-UX system.

1. To write the two files `mud.p` and `shark.f` to the tape (/dev/rmtb), type the following lines:

   ```
   cpio -ocv > /dev/rmtb
   mud.p
   shark.f
   Ctl-D (end of file)
   ```

   The options following `-o` may vary, depending on your particular situation. The *HP-UX Reference* details what options are available.

2. To read the tape, mount the tape on the system you are moving the files to and type:

   ```
   cpio -icv mud.p shark.f < /dev/rmtb
   ```

It is best to use relative path names (file names that **do not** start with /) when using *cpio*, so you can create a new directory to copy the files into. This will prevent you from writing over files with the same names on the destination system.

In order to read in a file saved by *cpio* you must know what options were used when the tape (or disc) was created. These same options should be used when restoring the files, except substitute **−i** for **−o**. We recommend that you always use the **c** option for portability. This option specifies that header information is to be written in ASCII. Note that *cpio* archives made with **−c** still contain a null character, so they cannot be electronically mailed. If directories need to be created when reading the tape to the destination system, use the **d** option.

Tcio is a pre- and post-processor for *cpio* that buffers data for a CS/80 data cartridge to reduce wear on the tape and tape head. To use *tcio*, simply pipe the output of *cpio* to it (when saving), or pipe the *tcio* output to *cpio* (when restoring). For instance:

```
ls | cpio -ocv | tcio -ov /dev/rmt
tcio -iv /dev/rmt | cpio -icv
```

The first command would create an archive of the current directory on the CS/80 tape. The second would restore that archive.

Dd is an alternative to *cpio* and *tar* that is covered under Trading Files with Other Systems.

Trading Files with Other HP Systems

HP-UX provides two simple ways of transferring files to or from other HP computers. If your HP-UX system and the HP system you wish to transfer your files to are both hooked up to an SRM (Shared Resource Manager), you can use the optional SRM access utilities to move the files between the systems via the SRM disc. If you do not have an SRM, you can use the LIF utilities to transfer your files. This more general situation is covered here.

LIF (Logical Interchange Format) is a Hewlett-Packard standard disc format supported by almost all HP computers. It is described under **LIF(1)** in the *HP-UX Reference*. HP-UX provides several utilities for manipulating LIF volumes and files.

There are some important things to know about LIF before using it as an exchange medium. First, the naming conventions for LIF files are different from those for HP-UX. LIF file and volume names are ten characters long, should be all uppercase and not contain any of the following characters:

```
#  *  ,  :  =  ?  [  ]  $  <  >
```

Keep these restrictions in mind when copying to a LIF volume. Additionally, the HP editors that use LIF files handle tab characters differently than you might expect, so you will have to run your files through *expand* before copying them into LIF form.

The following examples explain how to use LIF utilities and floppy discs to exchange files between an HP-UX system and an HP system that uses LIF.

From HP-UX to LIF

Adding Files to a LIF Volume

You can copy HP-UX files directly to a disc using *lifcp*, but it must have been previously initialized and have a LIF volume header written on it. The initialization routine is:

- The SYSTEM_MI bootable system on Series 200 HP-UX. (See the Media Initialization section of the *System Administrator's Manual.*)
- *Sdfinit*(8) on Series 500 HP-UX
- MEDIAINIT on HP Series 200 Pascal Workstations
- INITIALIZE on HP Series 200 and 500 BASIC Systems

After initialization, *lifinit*(1) is the HP-UX command for writing LIF volume headers.

The command:

```
lifcp hpux_file /dev/rfd0:FILE
```

translates `hpux_file` into LIF format and writes it to the flexible disc in disc drive 0. `FILE` is the name given to the file added to the disc. The previous contents of the disc are unchanged. If there is not enough room on the disc for the file, *lifcp* returns an error message.

If you are moving more than one file from HP-UX to a LIF volume on a relatively slow mass storage device such as a flexible disc, the process will be faster if you:

1. Create a LIF volume on the HP-UX file system.
2. Use *lifcp* to copy all the desired files into this volume.
3. *Cat* the volume to the disc.

Creating a LIF Volume

The command

```
lifinit -v270336 -d240 -nVOL VOLFILE
```

creates a LIF volume `VOL` in an HP-UX file `VOLFILE`. The option `-v270336` specifies the size in bytes of a 5 ¼ inch mini disc.

Note that the name of the LIF volume (VOL) is in all caps – this is a LIF standard.

Although LIF volume files can exist without problems on HP-UX, the system sees them as possible bad files and may generate a warning about them during execution of *fsck* (file system check). This does not mean that there is anything wrong with these files, only that HP-UX sees them as not strictly kosher.

Copying HP-UX Files to LIF Volumes

Once you have created a LIF volume, you must copy the files to the volume with *lifcp*. If the LIF volume file is `VOLFILE` use the command

```
lifcp hpux_file VOLFILE:FILE
```

to copy `hpux_file` to the LIF file `FILE`.

You will receive an error message if there is not enough room on the LIF volume for the file.

Moving the LIF Volume to Disc

When all of the files are written to the LIF volume file, *cat* the volume to your floppy disc.

1. Insert the disc into the drive.
2. Check the current contents of the disc, since step 3 will overwrite it. Do this with

   ```
   lifls /dev/rfd0
   ```

   where `/dev/rfd0` is the path name of the disc drive.
3. *Cat* the LIF volume file to the disc.

   ```
   cat VOLFILE  > /dev/rfd0
   ```
4. Remove the volume file from the current directory.

   ```
   rm VOLFILE
   ```

HP-UX to LIF Shell Script

The process described above can be automated with the following shell script. It performs all the necessary actions (including expanding tabs) except writing the volume file to the floppy disc. This step is omitted to allow use of different devices. The script assumes:

- The files will fit into a disc-sized volume.
- File names are all caps for LIF compatibility.

If these requirements are not met, *lifcp* will write an error message to *stderr*.

```
if [ $# = 0]
then
      echo "usage: liffiles FILE1 [FILE2 ...]" > &2
      exit 1
fi

if test -s VOLFILE
then
      rm VOLFILE
fi

lifinit -v270336 -nVOL -d240 VOLFILE

for i
do
       expand $i > tempfile
       lifcp tempfile VOL:$i
       echo "$i copied"
       rm tempfile
done

echo "finished copying files -- now cat VOLFILE to disc"
```

Shell Script for Copying to LIF Files

Create this file under the name `liffiles` and change its mode so that it can be executed.

```
chmod +x liffiles
```

Now you can copy a number of HP-UX files to a disc by executing

```
liffiles FILE1 FILE2 FILE3
cat VOLFILE > /dev/rfd0
```

Just list the names of the files you wish to copy on the command line, and change the device named in the *cat* command line to your particular drive.

From LIF to HP-UX

Copying LIF files to HP-UX is straightforward, but there are a few things to be aware of:

- The LIF files to be copied must be ASCII files. For instance, if you were moving files created on the Series 200 Pascal editor, you would have to translate these files from .TEXT to .ASC before attempting to *lifcp* them to HP_UX. If you get the error `CONFLICTTYPE` when you try to *lifcp* them, then the LIF files are not in the correct form.
- The actual name of your file on the LIF volume may be different than the name you used to reference it on your HP system. If the file is a .ASC file, the LIF name would be the name of the file (without .ASC) followed by the letter A and enough underscores to make the name ten characters long. Use *lifls* to see exactly how to reference it for *lifcp*.

To copy LIF files from a floppy disc to an HP-UX directory:

1. Place the disc in the drive.
2. Use *lifls* to list the contents of the disc

```
lifls /dev/rfd0
```

3. Copy the file `TESTA_____` from the disc to HP-UX file `test.c`.

```
lifcp /dev/rfd0:TESTA_____ test.c
```

Trading Files with Other Systems

The *dd* command can be used to transfer files from a general computer system to an HP-UX system or vice versa. This is a general purpose command for reading from and writing to mass storage devices. It is designed for use with 9-track tape drives, though it can be used with any supported mass storage device. Additionally, the *mt* command is used to position a tape for reading or writing. *Mt* can only be used with 9-track tape drives.

9-track tape that is unlabeled with ASCII records and is 1600 bpi phase encoded (7970E) is preferred. If the tape is blocked, you must know the block size (physical record length) and the blocking factor (logical record length). The maximum physical record length is 32 768 bytes.

Dd allows you you to specify any of these conversions:

- Change of blocking factor
- EBCDIC to ASCII
- Byte swapping. Bytes of memory are arranged in pairs and for HP-UX computers the byte with the higher physical address is logically the low order byte. The computer that produced your tape may not follow this convention.

Before using *dd*, you may need to re-position the tape so that unwanted files are skipped. *Mt* allows you to give directions to the tape drive to:

- Space forward over files or records
- Space backwards over files or records
- Write end-of-file marks
- Rewind the tape

The wide variety of tapes you might encounter makes it impossible to present an all-encompassing example for *dd*, but hopefully the following will generally illustrate its use.

```
dd if=/dev/rmt0 of=newfile ibs=800 obs=80 conv=ascii
```

This command would read an EBCDIC tape blocked ten 80-byte card images per record into the ASCII file `newfile`.

Using Datacomm to Move Files

If the systems you wish to transfer files between are both connected to a modem, then the files can be moved without using a temporary mass storage volume. The command you use to send files over the communications link depends on the types of the two systems.

UNIX to UNIX

There are several methods for transferring files between two UNIX systems. Which one you use depends on the configuration of your system, and your own personal preference. This section is not intended to be an all-inclusive reference; it presents the commands that are available, and general guidelines to their use.

UUCP

Uucp is a command for copying files from one UNIX system to another. It is a spooler – the files you wish to copy are placed in a directory on the local system. When the local system calls up the destination system, it transmits the file, along with the information needed to put the file in the proper place.

The syntax for *uucp* is the same as *cp* with the addition that file path names can be preceded by a system name. The file names for *uucp* have the form:

```
system-name!path-name
```

`system-name!` is optional. If neither the source or destination file name contain a system name, then *uucp* works exactly like *cp*. The system name must be one that is known to the local system (appears in the file /usr/lib/uucp/L.sys). You can get a list of the known systems with the command *uuname*. If you want to get your local system acquainted with some systems it doesn't already know, see your system administrator about adding entries to L.sys.

Example: To transfer all the Pascal source files (.p files) in your current directory from your local system to the system `vlsu-cs`, execute the command:

```
uucp -m *.p vlsu-cs!/usr/spool/uucppublic
```

The **−m** tels *uucp* to send you mail when the copy has been made. Note that c-shell users will have to precede the ! with \. For security reasons, you might not be able to use *uucp* to copy files from everywhere on the system. If you have problems, see your system administrator to find out what is allowed.

CU (Call Up)

Most UNIX systems with autodial modems provide *cu* (call up) to remotely access other UNIX systems. Once you have remotely logged in to a system, you can move files between it and your local system. Note that this method requires that you have a user id on both systems.

For example, suppose you have a file `greasy.f` on your UNIX system that you want to move to your new HP-UX system. There are two ways to use *cu* to do this. You can call up the UNIX system from your HP-UX system or vice versa. To perform the former operation, login to HP-UX and type:

```
cu -s1200 9=5558649
<login sequence to UNIX system>
~%take greasy.f
~.
```

The `-s1200` option specifies a 1200 baud modem; `9=5558649` tells the modem to dial 9, wait for a dial tone, and call 555-8649 (the phone number of the UNIX system). If you have immediate access to an outside phone line, omit `9=`.

If you had called up HP-UX from UNIX, then you would have typed

```
~%put greasy.f
```

to transfer the file to your HP-UX system. To give the file a new name (in case there's already a file of that name on the system you're moving to), put the new name after the old one like:

```
~%put greasy.f hpux_greasy.f
```

There is more information on *cu* in the *HP-UX Reference*.

Moving Files from Local Storage

You may have files that you've created on a personal computer that you want to move to your HP-UX system. If you have a modem and terminal emulation software on the personal computer then you should be able to transfer the files over a datacomm connection. The exact procedure depends on your particular computer and communications software, but here's an example that shows what HP-UX commands you would use.

1. Use your personal computer to login to HP-UX through a dial-up connection.
2. Type

   ```
   cat > hpux_file
   ```

   Any data that is sent to HP-UX after this command and before an end-of-file marker (Ctl-D) will be put into `hpux_file`.
3. Execute the command on your personal computer that sends the file you wish to transfer over the communications link.
4. Terminate the *cat* with Ctl-D (end-of-file). This step may not be necessary (depending on your particular communications program).

Cat may not behave as expected if the file being transferred contains special characters. To transfer such files, replace step 2 above by calling the *vi* editor as follows:

```
vi hpux_file
i (to get into insert mode)
```

When the transfer is done, hit the escape key to get out of insert mode and `ZZ` to exit the editor.

Subject Index

a

b

c

d

e

f

g

h

i

k

l

m

n

o

p

r

s

t

u

V

W

Manual Comment Card

If you have any comments or questions regarding this manual, write them on this comment card and place it in the mail. Include page numbers with your comments wherever possible. If there is a revision number, (found on the Printing History page), include it on the comment card. Also include a return address so that we can respond as soon as possible.

HP-UX Portability Guide
for the HP 9000 Series 200/500

98680-90045 October 1984

Update No. _____
(See the Printing History in the front of the manual)

Name: _____

Company: _____

Address: _____

_____

Phone No: _____

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
Fort Collins Systems Division
Attn: Customer Documentation
3404 East Harmony Road
Fort Collins, Colorado 80525