

MPE/iX POSIX Compliance Testing Using Lint Scanner

HP 3000 MPE/iX Computer Systems

Edition 1



**Manufacturing Part Number: B2476-90004
E1191**

U.S.A. November 1991

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 material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

UNIX is a registered trademark of The Open Group.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

© Copyright 1990, 1992 by Hewlett-Packard Company

- 1. Overview**
- 2. Hardware and Software Requirements**
- 3. Getting Started**
 - Contents of Tape Archive 10
 - MPE/iX POSIX Header Files 10
 - ANSI C Lib Header Files 10
 - MPE/iX POSIX Library Files 11
 - Installing the MPE/iX POSIX Library Files 12
- 4. Using the MPE/iX Library Standard with Lint**
 - Testing Your Application Against the MPE/iX POSIX Standard 14
 - What to Do with the Results 15
 - Limitations 16
- 5. Interpreting the Result**
 - Output Examples 20
 - Example 1 20
 - File-Specific Data 20
 - Summary Data 20
 - Example 2 21
 - File-Specific Data 21
 - Summary Data 21
 - Guidelines for Data interpretation 23

This document describes how to use the UNIX lint utility to test ANSI C applications for MPE/iX POSIX compliance. It is intended for use by UNIX application developers who are evaluating porting their applications(s) to the HP 3000 MPE/iX platform. Output from this test should be forwarded to the following address for review and interpretation by the appropriate HP representative.

All customers should send their output, as well as the name and number of a technical contact, to the following address:

Hewlett-Packard
Commercial Systems Division
19447 Pruneridge Ave.
Cupertino, CA 95014
Attn: MPE/iX POSIX Scanner Results

NOTE

Please send your output file via the media on which you received the original scanner files from HP. This will help us continue to provide this service at minimal cost.

Hardware and Software Requirements

While no specific hardware requirements exist, the user must be running a UNIX based operating system with a working `lint` utility that supports both the `-Aa` and the `-I` options, or their equivalents. These options are defined as follows:

- `-Aa` Invoke `lint` in ANSI mode
- `-I` Specify search path for `#include` <> files

Also required are two archive files, `mpeinclude.tar` and `mpelib.tar`, which contain the files defining the MPE/iX POSIX standard. You should have received these files on the tape entitled, *MPE/iX POSIX Declarations for Use with Lint Scanner*, in conjunction with this documentation.

Applications to be tested for compliance must be ANSI C compliant.

You should have received a tape archive entitled, *MPE/iX POSIX Declarations for Use with Lint Scanner*.

Contents of Tape Archive

This documentation includes two archive files, `mpeinclude.tar` and `mpelib.tar`. These archive files are comprised of the MPE/iX POSIX header files, ANSI C library header files, an MPE/iX POSIX library file, and an MPE/iX POSIX library declarations file.

MPE/iX POSIX Header Files

The following files are included in the `mpeinclude.tar` archive file:

- `/sys/stat.h`
- `/sys/types.h`
- `/sys/wait.h`
- `time.h`
- `limits.h`
- `dirent.h`
- `signal.h`
- `unistd.h`
- `errno.h`
- `fcntl.h`

ANSI C Lib Header Files

The following files are included in the `mpeinclude.tar` archive file:

- `assert.h`
- `float.h`
- `param.h`
- `stddef.h`
- `ctype.h`
- `times.h`
- `local.h`
- `stdlib.h`
- `malloc.h`
- `string.h`
- `values.h`

- `math.h`
- `stdarg.h`
- `varargs.h`
- `setjmp.h`
- `stdio.h`

MPE/iX POSIX Library Files

The following files are included in the `mpelib.tar` archive file:

- `posix.c` (source declarations file)
- `llib-lposix.ln` (library file)

Installing the MPE/iX POSIX Library Files

Follow the instructions below to install the MPE/iX POSIX archive files onto your system.

- Step 1.** Restore the two tar files from the tape entitled. *MPE/iX POSIX Declarations for Use with Lint Scanner*, into a working directory on your system.
- Step 2.** Make two new directories called `mpeinclude` and `mpelib`:
- ```
:mkdir mpeinclude
:mkdir mpelib
```
- Step 3.** Change directory to `mpeinclude`:
- ```
:cd mpeinclude
```
- Step 4.** Assuming that `mpeinclude.tar` is in the parent directory, then unarchive these files:
- ```
:tar xvf ../mpeinclude.tar
```
- Step 5.** Change directory to `mpelib`:
- ```
:cd ../mpelib
```
- Step 6.** Assuming that `mpelib.tar` is in the parent directory, then unarchive the files:
- ```
:tar xvf ../mpelib.tar
```

---

## Using the MPE/iX Library Standard with Lint

The following is a preparation checklist of items that may need attention before attempting to test your application against the MPE/iX Library Standard:

1. Be sure that your application is ANSI C compliant before attempting to test against the MPE/iX Library.
2. Be sure that your files are compiled and free of errors before attempting to check for POSIX compliance.
3. Check your application for `#include <>` statements. If these exist, then you will need to use the `-I` option when running `lint` with the MPE/iX POSIX Library.

---

**NOTE**

If your application is not ANSI C compliant or you do not have an equivalent to the `lint -I` option, you should contact your HP representative for an alternate means for MPE/iX POSIX compliance testing.

---

## Testing Your Application Against the MPE/iX POSIX Standard

The following is the syntax for invoking `lint` with the necessary options in order to test your application against the MPE/iX POSIX standard:

```
:lint -hanvb -Aa -I/mpeinclude - I /yourhdrdir \
-D_POSIX_SOURCE *.c /mpelib/llib-lposix.ln > result
```

Where:

|              |                               |                                                                                                                                                                                                          |
|--------------|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameters   | <i>*.c</i>                    | Application source files.                                                                                                                                                                                |
|              | <i>-hanvb</i>                 | lint options; see descriptions below.                                                                                                                                                                    |
|              | <i>/mpeinclude</i>            | HP-supplied MPE/iX POSIX header files.                                                                                                                                                                   |
|              | <i>/yourhdrdir</i>            | The path for any <code>#include &lt;&gt;</code> header files.                                                                                                                                            |
|              | <i>-D_POSIX_SOURCE</i>        | MPE/iX POSIX flag.                                                                                                                                                                                       |
|              | <i>/mpelib/llib-lposix.ln</i> | HP-supplied MPE/iX POSIX library file.                                                                                                                                                                   |
|              | <i>result</i>                 | The output file.                                                                                                                                                                                         |
| lint Options | <i>-h</i>                     | Do not apply heuristic tests that attempt to intuitively find bugs, improve style, and reduce waste.                                                                                                     |
|              | <i>-n</i>                     | Do not check for compatibility with either the standard or the portable lint library.                                                                                                                    |
|              | <i>-a</i>                     | Suppress messages about assignment of long values to variables that are not long.                                                                                                                        |
|              | <i>-b</i>                     | Suppress messages about break statements that cannot be reached.                                                                                                                                         |
|              | <i>-v</i>                     | Suppress messages about unused arguments and functions.                                                                                                                                                  |
|              | <i>-Aa</i>                    | Invoke lint in ANSI mode.                                                                                                                                                                                |
|              | <i>-I</i>                     | Change the algorithm for searching for <code>#include &lt;&gt;</code> files whose names do not begin with “/” to look in <code>mpeinclude</code> before looking in the directories on the standard list. |

## What to Do with the Results

Once the lint utility has been run using your application with the MPE/iX library files, then the output file, result, should be forwarded to the appropriate HP representative. This representative will then review the results and contact you for a more detailed discussion concerning the effort of porting your application to the MPE/iX platform.

You may, if you choose, refer to the following chapter on hints for doing some evaluation tasks on your own. In order to obtain a thorough and accurate evaluation of your results, however, you must submit your output file to your Hp representative for review.

All customers should send their output, as well as the name and number of technical contact, to the following address:

Hewlett-Packard  
Commercial Systems Division  
19447 Pruneridge Ave.  
Cupertino, CA 95014  
Attn: MPE/iX POSIX Scanner Results

---

### NOTE

Please send your output file via the media on which you received the original scanner files from HP. This will help us continue to provide this service at minimal cost.

---

## Limitations

The following are limitations to be considered when testing your application for MPE/iX POSIX compliance via the `lint` utility.

1. `lint` can only check C source files with the suffix `.c` or `.ln`.
2. The `-I` option of `lint`, or its equivalent, must be used to specify the search path for any `#include <>` files. If it is not used, then `lint` will not find these files and report them as non-POSIX compliant.
3. The application should be ANSI C standard, and the `-Aa` option of `lint` should be used in order to test for compliance.
4. The following C header files are defined in the POSIX 1003.1 standard, but are not supported for the first release of MPE/iX POSIX:

- `<grp.h>`

Contains the structure definition for an entry from the group database as returned by the function `getgrnam()`.

- `<pwd.h>`

Contains the structure definition for an entry from the user database as returned by the functions `getpwuid()` and `getpwnam()`.

- `<tar.h>`

Contains the symbolic constants used in the tar header block.

5. The following POSIX functions are defined in the POSIX 1003.1 standard, but are not supported for the first release of MPE/iX POSIX:

- `access`
- `ctermid`
- `dup2`
- `execle`
- `execvp`
- `execve`
- `execvp`
- `fpathconf`
- `getgroups`
- `pathconf`



- pipe
- setpgid
- setsid
- sysconf
- tcgetpgrp
- tcsetpgrp
- ttyname

If these functions are called, `lint` will report them as undefined.



This chapter provides some hints for interpreting the data that is obtained from the compliance testing using the `lint` utility. The output file may be very large, but this should not be cause for alarm, since much of the data will be duplicative in nature. For evaluation purposes, summary information provided by your HP representative upon review of output should be sufficient. The rest of the data should prove useful as a reference should you decide to port your application.

The output file will be structured so that file-specific data is presented first, followed by a summary of findings by category. This chapter will present examples of each and then provide some ways of filtering out the data that will be useful in evaluation.

---

**WARNING**

**General guidelines for interpreting results are given in this chapter. All results, however, should be reviewed with the appropriate HP representative to ensure that a thorough and accurate evaluation of porting effort is obtained.**

---

## Output Examples

The following are examples of output from the file-specific and the summary sections of the `lint` report.

### Example 1

This output is the result of `lint` run against a very small application coded so that it was non-conformant. The purpose here is just to illustrate a simple example of output format.

#### File-Specific Data

The following output sample is specific to the file, `xyz.c`. The first line states that in line 2 of `xyz.c`, `lint` was unable to locate the include file referenced, in this case `xxx.h`. The lines that follow flag an undefined variable, `i`, and present several warnings. The user should fix the undefined variable, but the warnings may be ignored.

```
xyz.c: 2: Can't find include file xxx.h
```

```
xyz.c
=====
(11) "i" undefined
(11) warning: "i" may be used before set
(12) warning: "path" may be used before set
(14) warning: "main" returns random value to invocation environment
=====
```

#### Summary Data

This output sample is the summary section printed out after the file-specific information. It has divided its messages into three categories:

- name used but not defined
- function argument ( ) used inconsistently
- function returns value which is always ignored

This information is useful for identifying unknown procedures, or procedures with calling sequences that differ from the MPE/iX POSIX standard.

```
 name used but not defined
 foo xyz.c(11)
 mktemp xyz.c(13)
function argument (number) used inconsistently
 chmod(arg 2) llib-lc(537) :: xyz.c(12)
function returns value which is always ignored
 chmod
```

## Example 2

This example presents actual data excerpts from the output file of running `lint` against a large, internal database application. This example illustrates the format of syntax errors, as well as numerous categories that may appear in the summary section.

### File-Specific Data

The following excerpt relates to the file, `BsCrType.c`. While the warnings may be ignored, the syntax errors should be investigated and corrected.

```

bs/BsCrType.c:
in.h(79) syntax error:
 struct { u_char s_b1, s_b2, s_b3, s_b4; } S_un_b;

in.h((80) syntax error:
 struct { u_short s_w1,s_w2; } S_un_w;

in.h(81) syntax error:
 u_long S_addr;

in.h(120) syntax error:
 u_short sin_port;

BsCrType.c
=====
(43) warning: "j" unused in function "BsCrTupleType"
(38) warning: "newtypekid" unused in function "BsCrTupleType"
(98) warning: "i" unused in function "BsCrSetType"
(94) warning: "lastcomptid" unused in function "BsCrSetType"

```

### Summary Data

The following excerpts are examples of the categories and summary information provided in the latter part of the output file.

```

name used but not defined
 UtWkspInit UtWksp.h(109)
 gethostname ClLogin.c(115)

name defined but never used
 callcode oaciEval.c(66)
 xllload xsaux.c(69)

```

```
name declared but never used or defined
 rresvport socket.h(247)
 OaciSetLastResult oaciEval.c(59)
```

<vex>

```
name multiply declared
 main BsMain.c(41) :: ClDrvr.c(890)
 DmiTempSecNum DmiGlobal.h(32) :: DmiGlobal.h(32)
```

```
value type used inconsistently
 strlen stdc.c(335) :: BsDelSec.C(130)
 strlen stdc.c(335) :: BsDelSec.c(131)
```

```
value type declared inconsistently
 buffer ClDrvr.c(78) :: sqlxdvr.c(418)
 lang oaciInit.c(31) : : OmFunStr.c(79)
```

```
function argument (number) used inconsistently
 memcpy(arg 1) stdc.c(274) :: BsBlobs.c(50)
 memcpy(arg 2) stdc.c(274) :: BsBlobs.c(50)
```

```
function used with a variable number of arguments
 strtod stdc.c(196) :: ClDrvr.c(311)
 strtod stdc.c(196) :: ClDrvr.c(318)
```

```
function value is used, but none returned
 OaciParseQuote OaciResetErr OmLogSt
 SvNewClient IqlInputHandler fdopen
```

```
function returns value which is always ignored
 InTupleAdd StDestroySemiCache OaciPreProcess
 InTypeGetChild tpparse OaciPrepareResult
```

```
function returns value which is sometimes ignored
 push_nargs InBagRemoveAt InTupleGetAt
 strcpy sprintf putchar
```

---

## Guidelines for Data interpretation

The following are guidelines for identifying potential portability problems in your application.

1. `grep` for “Can’t find include file” in the output file. For example:

```
:fgrep -n “Can’t find include file” result
```

Here, we used the output file, `result`, from Example 1 and obtained the following:

```
1:xyz.c: 2: Can’t find include file xxx.h
```

`fgrep` finds the pattern on line 1 of `result`. If `xxx.h` is your file, then you must use the `-I` option of `lint` to specify the location of the include directory and run `lint` again.

2. `grep` for “syntax error” in the output file. For example:

```
fgrep -n “syntax error:” result
```

Here, we used the output file, `result`, from Example 2 and obtained the following:

```
2:in.h(79) syntax error:
5:in.h(80) syntax error:
8:in.h(81) syntax error:
11:in.h(120) syntax error:
```

`fgrep` finds several occurrences of this string in the output file (lines 2, 5, 8 and 11). The syntax errors all occur in the file, `in.h`. These errors may be caused by missing include files or incompatibilities between MPE/iX and your UNIX system. These should be investigated and corrected before attempting to port.

3. `grep` for “undefined” in the output file. For example:

```
:fgrep -n “undefined” result
```

Here, we used the output file, `result`, from Example 1 and obtained the following:

```
5:(11) “i” undefined
```

`fgrep` finds one occurrence of this string in the output file in line 5. The error occurred on line 11 of some file. To find which file has the error, look in the output file, line 5, and see what the current file is. This error should be corrected, and `lint` should be run against the application once it has no compilation errors.

4. `grep` for “too many errors” in the output file. For example:

```
:fgrep -n “too many errors” result
```

Here, we used the output file, result, which appears as follows:

```
BsInitNo.c
=====
(27) "NameToOiddefArray" undefined
(27) type error in array expression
(27) operand for the indirection operator "*" should have non-void
pointer type
(27) struct/union name required before "."
(27) type error in array expression
(27) operand for the indirection operator "*" should have non-void
pointer type
(27) warning: "NameToOiddefArray" may be used before set
(30) "NameToOiddefArray" undefined
(30) type error in array expression
(30) operand for the indirection operator "*" should have non-void
pointer type
(30) struct/union name required before "."
(30) address operator "&" should not be applied to this operand
(30) struct/union name required before "."
(30) left-hand side of "=" should be an lvalue
(30)
**** cannot recover from this error ****
too many errors
```

The output from the grep is:

```
23:too many errors
```

fgrep finds one occurrence of this string in the output file in line 23. The errors that are listed for this file should be corrected and lint should be run again.

---

|             |                                                                                                                                                                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NOTE</b> | lint will stop checking a source file if it encounters a certain number of errors; the limit is system-specific. To obtain more accurate evaluation results, the errors noted for the file should be corrected, and lint should be run again. |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

5. grep for "cannot recover from earlier errors" in the output file. For example:

```
:fgrep -n "cannot recover from earlier errors" result
```

Here, we used the output file, result, which appeared as follows:

```
cl/ClLogin.c:
in.h(79) syntax error:
struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;

in.h(80) syntac error:
struct { u_short s_w1,s_2; } S_un_w;

in.h(81) syntax error:
u_long S_addr;
```



```
in.h(120) syntax error:
u_short sin_port;

socket.h(79) syntax error:
u_short sa_family; /* address family */

socket.h(85) syntax error:
u_short sp_family; /* address family */

ClLogin.c
=====
```

```
(234) struct/union "sockaddr_in" does not contain member "sin_port"
(234) left-hand side of "=" should be an lvalue
(235) struct/union "sockaddr_in" does not contain member "sin_addr"
(235) address operator "&" should not be applied to this operand
(236) cannot recover from earlier errors: goodbye!
```

The output from the grep is:

(236) cannot recover from earlier errors: goodbye!

fgrep finds one occurrence of this string in the output file in line 236. Look in the output file to see what file this corresponds to and correct the errors noted. Then run lint again.

6. grep for the following categories to locate summary information within the output file identifying potential compatibility problems:

- name used but not defined

Function names listed here will be non-POSIX according to the MPE/iX standard

- value type used inconsistently

For example:

```
strlen stdc.c(335) :: BsDelSec.c(130)
```

This example notes that the function, strlen, defined in stdc.c, line 335, is used in an inconsistent manner in BsDelSec.c, line 130, according to type.

- value type declared inconsistently

For example:

```
lang oaciInit.c(31) :: OmFunStr.c(79)
```

This means that lang is declared differently in oaciInit.c, line 31, than in OmFunStr.c, line 79.

- function argument (number) used inconsistently

For example:

```
function argument (number) used inconsistently
chmod(arg 2) posix.c(106) :: xyz.c(12)
```

This means that `chmod` is defined in `posix.c`, line 106, and called by `xyz.c`, line 12. You can compare your use of the function with the actual declaration found in `posix.c` in the `/mpe.lib` directory.

- function used with a variable number of arguments

For example:

```
strtod stdc.c(196) :: ClDvr.c(318)
```

This means that the call to the `strtod` function uses a different number of arguments in `ClDvr.c`, line 318, than as defined in `stdc.c`, line 196.

See **Example 2**, in the examples section of this chapter for other categories that are output, but of no particular use in identifying portability problems.