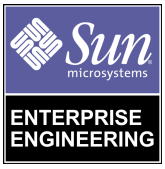
Operating Environments: Building Longevity into Solaris™ Operating Environment Applications

---

By Computer Systems, Solaris Productization and Marketing

Sun BluePrints™ OnLine - April 2000



http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-5334-10
Revision 01, April 2000

Operating Environments: Building Longevity into Solaris™ Operating Environment Applications

Sun releases new versions of the Solaris™ Operating Environment on an on-going basis, incorporating features and functionality in every release. Yet when each new Solaris Operating Environment release becomes available, customers wrestle with a troublesome question: if I upgrade to take advantage of new features, will my other applications still continue to work?

The answer is likely to be “yes” if those applications adhere to the Solaris Operating Environment application binary interfaces (ABIs). Sun defines the Solaris Operating Environment ABIs for this reason — the ABIs act as a contract, allowing developers to access operating system functionality in a way that Sun will support in future Solaris Operating Environment releases. The ABIs are the binary implementations of the supported application programming interfaces (APIs), and enable Sun to offer binary compatibility between different releases of the Solaris Operating Environment. When applications comply with the Solaris ABIs — for either SPARC™ processors or Intel — then they can be migrated seamlessly to the next Solaris Operating Environment release.

There are clear advantages to adhering to the Solaris ABIs/APIs in writing new applications and maintaining existing applications. For developers, sticking to the standard APIs and dynamic linking reduces the need for application maintenance in the future — developers are not necessarily faced with producing a new application release to match each new Solaris Operating Environment release. For end-users, this means that a new Solaris Operating Environment release — such as the Solaris 8 Operating Environment release — can be installed and existing applications will still work, giving them access to desirable new features and more powerful Sun hardware platforms.

This article focuses on specific steps that developers can take to improve the longevity of their applications. It discusses the use of standard interfaces, static and dynamic linking, Sun's new tools for testing ABI compliance, and some evolutionary new features and interfaces that Sun is specifically offering in the Solaris 8 Operating Environment release.

Steps for Improving Longevity in Solaris™ Operating Environment Applications

To build greater longevity into Solaris Operating Environment applications, developers should, most importantly, follow these two steps:

1. Write applications based on standards (e.g., POSIX, MOTIF, X11), the standard Solaris APIs, and their public interfaces.
2. Link to dynamic rather than static libraries.

What are the Standard Solaris Operating Environment APIs?

The standard Solaris Operating Environment APIs are documented in the System Calls and Library Functions (sections 2 and 3) of the man pages. These are the standard, public means for applications to access operating system functionality. See the *Solaris 8 Reference Manual,* available in AnswerBook on-line documentation, or view *http://docs.sun.com*.

A common short-cut among developers is to access system database files directly, rather than using the corresponding library functions. System database files, which are documented in Section 4 of the man pages, are subject to change from one Solaris Operating Environment release to the next. For example, the following code:

```
fp = fopen("/var/adm/utmp", "r")
```

may function on a Solaris 2.5 Operating Environment system, but it will fail on a Solaris 8 Operating Environment system. In the Solaris 8 Operating Environment, the system database file `/var/adm/utmp` has been superseded by `/var/adm/utmpx`.

If the library function `getutxent()` had been used instead, then the application would successfully obtain information from the appropriate system database file, regardless of the operating environment.

C Compilation Tips

To promote more stable and compatible applications, developers can specify the "`-Xa`" and "`-v`" compiler flags. The "`-Xa`" option indicates that the Sun C compiler will issue warnings when the code contains constructs for which K&R C and ANSI C specify different semantics (the compiler uses the ANSI C interpretation, which results in more portable code). The "`-v`" option directs the compiler to perform more strict semantic checks and enables other lint-like checks, such as detecting functions that lack return values. These options can often help developers avoid problems, resulting in applications with greater stability and fewer defects.

Using Dynamic Linking

To develop applications compatible with future Solaris Operating Environment releases, it is critical to *link to dynamic rather than static libraries*. Each release of the Solaris Operating Environment provides both static and dynamic libraries. Static libraries build symbol dependencies on the current operating environment into the application. The library interfaces are the same, but the implementations of the shared objects may be different from operating system release to release. By linking an application with static libraries, dependencies on the operating system become "bolted" into the application. When the application is executed on a later operating system release, it may fail because of these hardwired dependencies.

For example, a TCP/IP client program might call `socket()` and `connect()` to connect to a network service. If that program is compiled on Solaris 2.4 Operating Environment and statically linked to libsocket.a, the program will execute on Solaris 2.4, 2.5, and 2.5.1 Operating Environments. However, if that executable is moved to a system with Solaris 2.6 Operating Environment or later, the `socket()` call will fail. In Solaris 2.6 Operating Environment , the underlying socket implementation changed from STREAMS-based sockets to in-kernel sockets. Applications linked to static libraries prior to version 2.6 that invoke socket library functions will have problems on Solaris 2.6 Operating Environment and above — only applications that dynamically link with the socket library will continue to run properly.

To avoid building these dependencies into applications, developers should primarily use dynamic linking. With dynamic linking, applications access shared objects at run-time — when the application is actually executed.

Isolating “Non-Standard” or Statically Linked Code

In some cases static linking cannot be avoided, which is why Sun still provides static libraries with every release. Applications code that performs authentication or other security-related functions, for example, should be statically linked to prevent run-time linking with substituted shared objects that could pose a security risk. Isolating code that requires static linking (or any other non-standard code) allows developers to focus on these code segments when porting to a new operating system release.

Testing Applications for Potential Problems

Sun provides several tools which can be used to test compliance with the Solaris Operating Environment ABI and help developers migrate Solaris Operating Environment applications:

- `appcert` — detects the use of non-standard interfaces or static linking.
- `apptrace` — traces library-level calls.
- `appcheck` — reports on the use of known problematic interfaces.

Of these, `apptrace` and `appcheck` are new in the Solaris 8 Operating Environment.

Before migrating an application to a new Solaris Operating Environment release, `appcert` can be used on executables to identify potential compatibility problems (Figure 1). `appcert` will highlight any issues, such as the use of private interfaces or static linking. The application can then be modified, recompiled, if necessary, or

relinked before testing the application on the target release. When the application is tested on Solaris 8 Operating Environment, `apptrace` and `appcheck` can be used to further troubleshoot application problems.

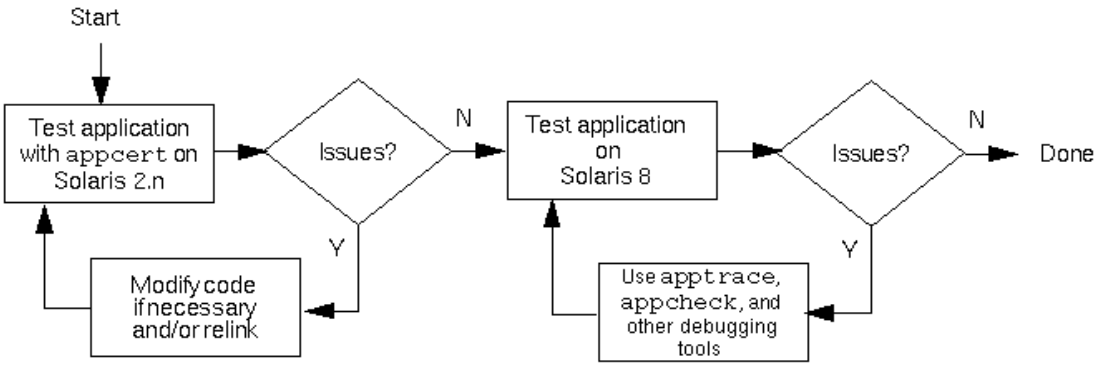


FIGURE 1 Using `appcert`, `apptrace`, and `appcheck` on Solaris™ Operating Environment releases

Using `appcert`

`appcert` consults databases containing the private and public symbols for each Solaris Operating Environment release (2.3 to 8), allowing applications to be tested for forward compatibility. `appcert` detects the following types of problems:

- *PRIVATE_SYMBOL_USE.* `appcert` flags the use of private symbols from Solaris Operating Environment libraries. These symbols are often the interfaces that the libraries use to call and communicate with one another.
- *DEMOTED_SYMBOLS.* Demoted symbols are Solaris Operating Environment library functions or data variables that were once private and have since been removed.
- *UNBOUND_SYMBOLS.* If the dynamic linker cannot resolve symbols when `appcert` is run, then `appcert` reports “unbound symbols.” This may occur if the LD_LIBRARY_PATH variable is not set correctly and the dynamic libraries cannot be found.
- *UNRECOGNIZED_SYMBOLS.* Unrecognized symbols are those which do not appear in `appcert`'s database of private and public symbols. This may indicate that a symbol is not present on the release that `appcert` is checking against.

- *NO_BINDINGS_FOUND.* On each binary object to be tested, `appcert` runs `/bin/ldd -r`. If this command fails (for example, the binary object does not have read permission or it is completely statically linked), `appcert` will have no dynamic symbol binding information and will report “no bindings.”
- *STATICALLY_LINKED.* `appcert` detects static linking of archives libc.a, libsocket.a, and libnsl.a (the corresponding shared objects were not dynamically linked).
- *COMPLETELY_STATICALLY_LINKED.* The checked binary has no dynamic dependencies at all.

Downloadable from *http://www.sun.com/developers/tools/appcert/* as free, unsupported software, `appcert` is a Perl script. Perl now ships as part of the Solaris 8 Operating Environment, but it is also included with the `appcert` distribution to simplify installation on previous Solaris Operating Environment releases.

`appcert` can be used by specifying the target release and the release on which the application was built, as shown in Figure 2. (Note that `appcert` can only approximate dynamic bindings when checking against releases other than the

specific release where `appcert` is executed.) `appcert` checks the specified executable, or a series of executables in a given directory. `appcert` also generates a summary report of all ABI symbols used (Figure 3).

```
% appcert -P ./bin/workman -b "Solaris2.6" -c "Solaris8" ./bin/workman

Appcert will profile and check all of the binary executables and shared object
libraries contained in your application or product. It checks for a number of
potentially unsafe practices that may lead to your application breaking on
future Solaris releases...

------------------------------------------------------------------------
Finding executables and shared libraries to check ...

Estimated space required for output: 117K
Profiling 1 binary objects ...
profiling: bin/workman
Checking binary objects for unstable practices ...
checking:  bin/workman
------------------------------------------------------------------------
Generating report ...

        ************************************************************
                            Sun ABI Program
                    Software Product Binary Stability Tool
                      (c) 1995-1999 by Sun Microsystems, Inc.

                    Interface Use Stability Report
                    Appcert version: 2.0 BETA
                    Generated by: "@(#)symreport 1.1   99/08/16 SMI"
                    Wed Feb 16 10:59:28 CST 2000

        ************************************************************
        Product Examined: ./bin/workman
        Platform Version Built On: SunOS 5.6 (Solaris2.6)
        Platform Version Checked:  SunOS 5.8 (Solaris8)

        Summary: Potential binary stability problem(s) detected.

A total of 1 binary objects were examined.

The following (1 of 1) components have potential stability problems,
    or otherwise could not be completely checked:

   bin/workman  (1 demoted (removed) private symbols: libc.so.1:_nss_XbyY_fgets)
------------------------------------------------------------------------
The above report has been placed in the file:
        /tmp/appcert.18841/Report

                (...and so on)
```

FIGURE 2 Example of using `appcert`

```
*****************************
ABI SYMBOL USAGE SUMMARY REPORT
*****************************

  Binary Object: bin/workman
  System: SunOS mtv 5.8 Beta_Refresh sun4u sparc SUNW,Ultra-1
  Release built for:         SunOS 5.6 (Solaris2.6)
  Checked against ABI model: SunOS 5.8 (Solaris8)
  Appcert Version: 2.0 BETA
  Generated by: "@(#)symreport 1.1   99/08/16 SMI"
  Date: Wed Feb 16 10:59:28 CST 2000
  By: gha@mtv

*********************************************
References to shared objects in the Solaris8 ABI
*********************************************

  /usr/lib/libc.so.1
  /usr/lib/libsocket.so.1
  /usr/openwin/lib/libxview.so.3

*********************************************
Symbol usage statistics (summary by shared object)
*********************************************

  Library                          Public  Private
  -------                          ------  -------
  /usr/lib/libc.so.1                   64        0
  /usr/lib/libsocket.so.1               1        0
  /usr/openwin/lib/libxview.so.3       38        0

*********************************************
Symbol usage (detailed inventory by shared object)
*********************************************

Symbols in /usr/lib/libc.so.1 Directly Referenced

  64 public symbols are used:
    .div       .mul       .rem       __ctype    __filbuf   __flsbuf
    __iob      _exit      access     atexit     atoi       calloc
    close      dup2       execl      execlp     exit       fchmod
    fclose     fcntl      fflush     fgets      fopen      fork
    fprintf    fputc      fputs      free       fscanf     fseek
    fstat      ftell      fwrite     getenv     getopt     getpid
    ioctl      kill       malloc     memcpy     memset     open
    optarg     perror     poll       printf     rand       realloc
    realpath   setitimer  sigaction  sprintf    srand      strcat
    strchr     strcmp     strcpy     strlen     strncmp    strrchr
    unlink     ustat      wait       write

Symbols in /usr/lib/libsocket.so.1 Directly Referenced

  1 public symbols are used:
    socketpair

                (...and so on)
```

FIGURE 3 Excerpt of an `appcert` summary report

For more information on `appcert` and the kinds of problems it detects, see the `appcert` man page (*/var/tmp/Appcert/man/appcert.1*). Sun is developing an extensive guide on using `appcert`, which will be available soon.

apptrace

In migrating applications to the Solaris 8 Operating Environment, `appcert` can be used to highlight compatibility problems, and then `apptrace` and `appcheck` can further troubleshoot applications. Bundled in Solaris 8 Operating Environment , `apptrace` traces calls to Solaris shared library routines during application execution. It is somewhat like `truss`, although `apptrace` traces library-level rather than system-level calls in the executable. `apptrace` reports the name of each library interface called, the values of the arguments passed, and provides return values:

```
workman  -> libc.so.1:atexit(func = 0xff3ca6e8) = 0x0
workman  -> libc.so.1:atexit(func = 0x44bd0) = 0x0
workman  -> libxview.so.3:xv_init(0x4a070802, 0xffbef3dc, 0xffbef3fc)
workman  -> libxview.so.3:xv_unique_key(0x5b038, 0xff07ea98, 0xffbeeef8)
workman  -> libc.so.1:getenv(name = "WORKMANRC") = "<nil>"
workman  -> libxview.so.3:defaults_get_string(0x57b18, 0x57b2c, 0x0)
workman  -> libc.so.1:getenv(name = "WORKMANDB") = "<nil>"
workman  -> libxview.so.3:defaults_get_string(0x57b40, 0x57b54, 0x0)
workman  -> libxview.so.3:defaults_get_integer(0x57b68, 0x57b84, 0x2)
workman  -> libc.so.1:strrchr(s = "./bin/workman", c = 0x2f) = "/workman"
workman  -> libc.so.1:strlen(s = "./bin") = 0x5
workman  -> libc.so.1:malloc(size = 0x10) = 0x62c10
workman  -> libc.so.1:sprintf(buf = 0x62c10, format = 0x44c28, ...) = 15
workman  -> libxview.so.3:defaults_get_string(0x57bac, 0x57bc0, 0x62c10)
workman  -> libc.so.1:getopt(argc = 0x1, argv = 0xffbef3fc, optstring =
"s:p:dc:ol:eXnbV:hCD:") = 0xffffffff errno = 0 (Error 0)

                          (...and so on)
```

Output from `apptrace`, `sotruss`, and `truss` can be valuable in tracking down a problem that occurs at run-time, although the output can be quite lengthy and time-consuming to analyze.

appcheck

`appcheck` — unsupported software that Sun is currently developing — captures calls to known problematic interfaces, examines their arguments, and looks for complications that may inhibit binary compatibility. Like `apptrace`, `appcheck` is execution-dependent — it traces only code that is actually executed. Because `appcheck` relies on a knowledge base of interfaces that are prone to problems, its output is less lengthy than that of `apptrace`, allowing developers to more easily zero in on typical problems. Like `appcert`, `appcheck` will be available to developers from the Sun web site.

Evolutionary Technology in Solaris 8 Operating Environment

With every release of the Solaris Operating Environment, Sun adds new features. At the same time, Sun is careful to preserve existing interfaces in order to maintain compatibility. Sun's goal is to provide revolutionary new technology in an evolutionary manner.

IPv6

Sun has added the Internet Protocol version 6 (**IPv6) capability** to support a broader range of IP addresses, in accordance with the IPv6 standard. At the same time, Sun has maintained full support of the existing IPv4 interfaces. As shown in Table 1, the new IPv6 interfaces, such as `getipnodebyname()`, will work with either IPv4 or IPv6-style addresses, so new applications can easily support both address types. Applications that use IPv4 interfaces, such as `gethostbyname()`, will continue to function on the Solaris 8 Operating Environment without the need for any changes.

IPv4 Interfaces	IPv6 Interfaces
`AF_INET`	`AF_INET6`
`sockaddr_in` structure for IPv4 address, e.g., returned by `getnameinfo()`	`sockaddr_in6` structure large enough for either IPv4 or IPV6 address, e.g., returned by `getnameinfo()`
`gethostbyname()` gets an IPv4 address for a given host	`getipnodebyname()` gets either an IPv4 or IPv6 address for a given host

TABLE 1 Comparison of IPv4 and IPv6 Interfaces in the Solaris 8 Operating Environment

IPv4 Interfaces	IPv6 Interfaces
`gethostbyaddr()` gets a hostname for a given IPv4 address	`getipnodebyaddr()` gets a hostname for either an IPv4 or IPv6 address
`inetaddr()` converts an IPv4 address in standard text presentation form into its numeric binary form	`inetpton()` converts either an IPv4 or IPv6 address in standard text presentation form into its numeric binary form
`inetntoa()` converts an IPv4 numeric address into standard text presentation form	`inetntop()` converts either an IPv4 or IPv6 numeric address into standard text presentation form

TABLE 1 Comparison of IPv4 and IPv6 Interfaces in the Solaris 8 Operating Environment

Alternate Thread Library

Also in the Solaris 8 Operating Environment, Sun has introduced an alternate thread library that uses a 1-level (rather than 2-level) model of mapping user threads to kernel LWPs. This library can have positive performance impacts for some applications, especially for some real-time applications that require more predictable scheduling. Applications can actually test performance impacts simply by relinking with the alternate thread library in `/usr/lib/lwp`. To do this, set the environment variable LD_LIBRARY_PATH:

```
LD_LIBRARY_PATH=/usr/lib/lwp (for 32-bit applications)
LD_LIBRARY_PATH_64=/usr/lib/lwp/64 (for 64-bit applications)
```

or the variable LD_PRELOAD:

```
LD_PRELOAD=/usr/lib/lwp/libthread.so (for 32-bit applications)
LD_PRELOAD=/usr/lib/lwp/64/libthread.so (for 64-bit applications)
```

The alternate thread library does not replace the 2-level thread library — it simply extends its functionality. Linking to an alternate library such as the alternate thread library, demonstrates the power of dynamic linking. Sun can provide alternate libraries as a means of enhancing performance for applications without having to introduce new interfaces.

/dev/poll

A new polling mechanism (/dev/poll) in the Solaris 8 Operating Environment provides higher performance when polling on a very large number of file descriptors, and supplements, but does not replace, poll(2). The /dev/poll driver is a special driver that allows multiple sets of polled file descriptors to be monitored. By using this driver, applications can poll large numbers of file descriptors very efficiently. Poll-intensive applications that use /dev/poll are likely to see a large performance boost. For more information, see the man page on poll(7d).

Modular Debugger

The Solaris 8 Operating Environment also provides a powerful new modular debugger (mdb) for analyzing kernel-level and user-level processes. Mdb is extremely useful for post-mortem core analyses, and can be especially useful for identifying memory leaks. For more information, see the *Solaris Modular Debugger Guide* and the mdb man page.

Software Installation Improvements

To simplify the installation of applications, developers should use the standard SVR4 package format as specified in the *Application Packaging Developer's Guide*. (Package format provides a standardized means of installing and maintaining system and application software.) The Solaris 8 Operating Environment also includes new installation features that can leverage the standard SVR4 package format:

- During the installation of the operating environment, new Solaris Web Start™ Wizards technology allows customers to specify what software should be installed. Developers can also use this Solaris Web Start Wizards technology to facilitate easy application installation. Soon, Sun will release a Software Development Kit to allow software providers to take advantage of Solaris Web Start Wizards functionality.
- A new product registry feature in the Solaris 8 Operating Environment keeps track of what software is installed via pkgadd or Solaris™ Web Start. The product registry, accessible via the prodreg command, provides an easy-to-use interface for administrators to track installed software, specific version information, installation dates, etc.
- A new installation CD kicks off the Solaris 8 Operating Environment installation process. OEM developers may wish to include applications on this CD to facilitate the easy distribution of the operating environment and applications. (In the future, Sun plans to release a Software Development Kit for developing customized installation CDs.)

For more information about new features in the Solaris 8 Operating Environment, see the guide *What's New in Solaris 8 (*in the on-line Solaris 8 Answerbook documentation or via *http://sun.docs.com),* or look for other Sun BluePrints™ OnLine articles.

Summary

In writing new applications and modifying existing ones, developers should use standard interfaces and link dynamically to ensure Solaris Operating Environment ABI compliance. For existing applications, `appcert` on the current Solaris Operating Environment release can be used for troubleshooting applications. Once the application has been moved to a Solaris 8 system, then developers can use other tools, such as `apptrace` and `appcheck`, to further isolate and resolve any remaining problems. Following these steps will provide binary compatibility and improve the longevity of applications on the Solaris Operating Environment.