Oracle Solaris Guarantee Program (valid for Oracle Solaris lifetime)

Oracle Solaris is designed and tested to protect customer investments in software.

While new functionality may be introduced in new releases, Oracle Solaris is designed with continuity of binary interfaces, so applications developed on earlier releases can continue to run. This enables customers to purchase new systems or upgrade the OS on older systems and continue to run their existing applications.

Customers and Partners who have purchased Oracle Premier Support for Operating Systems can receive assistance in resolving compatibility issues identified when moving a binary application from an earlier OS release.

For developers, the OS presents an architecture-neutral API meaning that a program developed on SPARC architecture can be recompiled to run on x86, and vice-versa.

See below for more details on the Binary and Source guarantees.

Oracle Solaris Binary Application Guarantee

The Oracle Solaris Binary Application Guarantee reflects Oracle's confidence in the compatibility of applications from one release of Oracle Solaris to the next and is designed to make re-qualification a thing of the past.

Binary compatibility between releases of Oracle Solaris helps protect your long-term investment in the development, training and maintenance of your applications:

A binary application built on Solaris 2.6 or later that makes use of operating system interfaces as defined in stability.7 run on subsequent releases of Oracle Solaris, including their initial releases and all updates, even if the application has not been recompiled for those latest releases.

If an application experiences a compatibility problem when running on your latest supported Oracle Solaris Operating System, support is offered as described below:

For Oracle Solaris 10

Use the integrated "appcert" utility (see the man page for appcert and the page on the Solaris ABI Tools) to check your application. If no errors are reported but problems running the application remain, a Service Request (SR) should be opened to obtain support.

For Oracle Solaris 11 and subsequent releases

The Preflight Application Checker tool is used to verify application compatibility and can be downloaded from: https://www.oracle.com/solaris/solaris11/downloads/solaris11-preflight-checker-tool-v113-downloads.html. The package includes documentation that describes how to check an application for compatibility with Oracle Solaris.

A Service Request should be opened (https://support.oracle.com/) if problems remain when running the application.

Some documented but public interfaces become obsolete or can change, such as those documented as Volatile stability instead of Committed or Uncommitted in their respective man pages. For more information, see the Interface Stability section of stability.7. Also, the End of Feature Notices for Oracle Solaris 11 page has the list of interfaces and features that were removed or are planned for removal in Oracle Solaris 11 updates.

Oracle Solaris Source Code Guarantee

Oracle makes it easy for developers to build, test and support their applications for deployment on both SPARC and x86 systems. With an OS built from a single source base, architecture neutral API and the Oracle Developer Studio tools offered on both architectures, Oracle helps your code be readily portable between the two platforms.

Oracle's Solaris Source Code Guarantee provides the confidence that if you develop and successfully compile C and C++ applications to run on SPARC or x86 platforms, these applications will compile and run on either of these platforms.

If you have an application that was developed on SPARC-based platforms, it can be recompiled to run on x64 platforms for the same release, using the same version of Oracle Developer Studio (and vice-versa). If the application meets the eligibility guidelines set out below and does not recompile, an SR should be opened to request assistance.

Specific Criteria for Source Compatibility:

1) General Source Code Guidelines for Eligible Applications:

- must be written to comply with the standards as listed in standards.7;
- must not be converting 32 bit applications to 64 bit applications or vice versa;
- must not include Makefiles for building object files from compiled applications to run on specific platforms;
- must not include 3rd party software libraries (binaries) or other programming interfaces;
- must not include Assembler code in any form.

2) Compiler Guidelines for Eligible Applications:

- must be C and C++ applications;
- must have been compiled with the same compiler and OS version on both the Originating Platform and the Target Platform;
- must have been compiled using a supported Oracle Developer Studio compiler on a supported OS release.

Third Party applications, including header files, need to be compiled with the same version of the compiler as the application itself.

The use of Compiler flags and pragmas (e.g. Compiler overrides and controls) must be compatible with the Instruction Set Architecture that you are using. See Oracle Developer Studio documentation for a list of compiler flag issues. (e.g., if your application has been optimized for the SPARC Platform using platform-specific compiler flags, some flags may be incompatible for the x86 platform, and you will need to delete these flags or replace them with the flags that are valid on the x86 platform and used for optimization for the x86 platform).

Applications with platform-specific definitions (for example, #ifdef __SPARC), may require your intervention before porting between platforms.

Performance-sensitive applications generally use platform-specific optimization flags to achieve good performance. These flags are architecture dependent and are different for the SPARC and x86 platforms. Certain compiler options work only for the SPARC platform and should not be used on the x86 platform and vice versa.

The Interfaces to I/O devices on the SPARC and x86 platforms must be compliant with the interfaces defined in the Solaris DDI and kernel interface man pages (section 9).

Floating Point Operation Differences – Computation results may differ between SPARC and x86 CPUs due to differences in floating point register size or library routines. Please refer to the Numerical Computation Guide for further detail.

Big Endian/Little Endian Guidelines for Eligible Applications – The Oracle Solaris Operating System runs on both big-endian mode SPARC and little-endian x86 systems. Endianness is mandated by the hardware platform used. One of the strengths of the Oracle Solaris OS is that it handles endianness such that ISVs or developers do not need to be concerned about architectural differences when writing code. The software architecture allows big-endian applications developed on the SPARC Platform to run on little-endian x86 systems when they are recompiled. In general, no re-engineering is needed. However, the endianness difference can cause problems, particularly if an application includes low-level code that directly addresses the hardware.

Shared Memory Implementation – Sharing memory with opposite-endian devices or processors constitutes data import and export. Access to shared data must take endianness issues into account. Some peripheral buses lend themselves to shared memory approaches to interprocessor communication. In general, some means of translating the data must be provided if the processors are not of the same endianness.

Application Data Storage – File systems are neutral to endianness in general, and swapping files between Oracle Solaris on SPARC and x86 is not an issue. But applications storing raw data that can be shared between platforms would be an issue. For example, if an application on the SPARC platform writes the data structures in a raw format to the files, then the data stored in these files would be endian-dependent. Reading or writing these data files from an x86 processor-based machine can create problems regarding the endianness of the data. In short, the binary data (raw data) stored on a file is not transferable between the SPARC and x86 platforms.

Storage Order Differences – The order of storage of the data varies between platforms; therefore, code written assuming a particular byte ordering (Big Endian vs. Little Endian) is not portable.

Accessing the Individual Bytes of the Numeric Data Using Pointers – If an application uses individual bytes of a numeric data type to store and retrieve values, then you might get different results due to the endianness of the underlying platform. In this case the code would not be portable between SPARC and x86 platforms.

Sending Data Over the Network – Applications running on Oracle Solaris systems interoperate with other systems over the network using standard network protocols that have enabled communication between big- endian and little-endian systems. However, if the applications transfer data (numeric data) over the network directly without any protocol, then an issue might arise.

3) Platform Guidelines for Eligible Applications:

I/O Architecture – Unlike the x86 family, which uses special IN/OUT instructions to access the PCI I/O space, the SPARC CPU family treats access to the PCI I/O space the same as it treats access to the memory space. Communication with I/O devices in the SPARC platform is accomplished through memory; a range of memory locations is logically replaced by device registers.

The x86 platform accesses I/O ports through the I/O address space by means of a set of I/O instructions and a special I/O protection mechanism. Accessing I/O ports through memory-mapped I/O is handled with the processor's general-purpose move and string instructions, with protection provided through segmentation or paging. I/O ports can be mapped so that they appear in the I/O address space or the physical-memory address space (memory-mapped I/O) or both.

The SPARC platform assumes that input/output registers are accessed by means of load/store alternate instructions, normal load/store instructions, coprocessor instructions, or read/write ancillary state register instructions (RDASR, WRASR). In the load/store alternate instructions case, the I/O registers can only be accessed by the supervisor. If normal load/store instructions, coprocessor instructions, or read/write Ancillary State Register instructions are used, then whether the I/O registers can be accessed outside of supervisor code or not depends on the implementation.

Must Not Use ISA-specific pathnames – must not use the instruction set architecture name in a pathname used by the Eligible Application.

Must Not Use ISA-specific interfaces – Use of SPARC specific interfaces have no equivalent in x86 or x64 architectures and will not compile (e.g. SPARC registers, SPARC traps, etc.).