Developer's Guide for Migrating to Oracle Solaris 11

Part No: E60338
April 2020

ORACLE

Developer's Guide for Migrating to Oracle Solaris 11

Part No: E60338

Référence: E60338

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Provides information about compilers, tools, and tips which can be helpful for migrating from any other UNIX platform to Oracle Solaris 11.
- **Audience** – Developers who want to migrate to Oracle Solaris from other UNIX platforms.
- **Required knowledge** – Basic experience in developing applications on UNIX-based platforms.

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E53394-01.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

About Migrating to Oracle Solaris

This chapter presents the advantages of migrating to Oracle Solaris 11 and the factors that you need to consider before migrating.

Advantages of Migrating to Oracle Solaris

Oracle Solaris 11 provides a large set of native (Oracle proprietary) and open-source commands, tools, libraries and platform services similar to those available on other UNIX-based platforms such as Linux, HP-UX and AIX.

The key advantages of Oracle Solaris 11 are as follows:

- Portable, scalable, interoperable, and compatible.
- Supports portability across different architectures.
- Conforms to standards for application portability.
- Supports tools optimized for Oracle hardware devices.
- Supports a wide range of hardware.
- Has infrastructure designed to scale on large systems without needing to touch the code.
- Supports IEEE Std 1003.1 and IEEE Std 1003.2, commonly known as POSIX.1 and POSIX.2, respectively. Specifically, Oracle Solaris 11 conforms to POSIX.1-2004. For more information, see `https://publications.opengroup.org/t101` and `http://pubs.opengroup.org/onlinepubs/9699919799/mindex.html`.
- Offers end-to-end 24 x 7 support from Oracle (a single vendor) for hardware to applications.
- Supports programming languages such as Perl, Ruby, and Python.
- Offers the Binary Guarantee Program that guarantees binary compatibility for your applications across various Oracle Solaris OS releases. For more information, see Oracle Solaris Guarantee Program (`http://www.oracle.com/us/products/servers-storage/solaris/solaris-guarantee-program-1426902.pdf`).
- Offers support for open-source software.

Note - Oracle Solaris 11 can also be used with other Oracle products to obtain an integrated stack (hardware to applications) optimized to provide an end-to-end process of application development and deployment. Using Oracle integrated systems ensures you use components tailor-made to function together to support your development efforts. For more information, see Engineered Systems Documentation (`https://docs.oracle.com/en/engineered-systems/`).

Considerations Before Migrating to Oracle Solaris

Before migrating to the Oracle Solaris OS, you must consider the following factors:

- Differences in file system format
- 32-bit and 64-bit architecture
- Endianness of the platform
- Support for the compiler options you need in Oracle Solaris
- Differences in threading models
- Build tools and other build dependencies such as `gmake`, `dmake`, `make`, and ANT
- Database configuration
- Migration effort involved
- Licensing costs and migration of licenses
- Support for open-source software in Oracle Solaris

For more information, see "Considerations and Dependencies for Migrating a Database to Oracle Solaris" on page 17.

♦ ♦ ♦ CHAPTER 2

Migrating the Developer Environment

This chapter provides information about migrating the developer environment to Oracle Solaris 11. It also discusses migrating data and databases.

This chapter contains the following topics:

Development Environment in Oracle Solaris

Oracle Solaris provides and supports various tools for application development. For information about the tools available for development, see *Setting Up the Application Development Environment in Oracle Solaris 11*.

Using Compilers in Oracle Solaris

Oracle Solaris supports the open-source compiler system GNU Compiler Collection (GCC), which is available for installation as an IPS package. You can also use Oracle Developer Studio instead of GCC for software development. Oracle Developer Studio is a comprehensive C, C++ and Fortran tool suite for both, Oracle Solaris and Linux operating systems. For more information about how to use Oracle Developer Studio, see "Optimizing Applications by Using Oracle Developer Studio" on page 25.

Using Oracle Developer Studio for Application Development

Oracle Developer Studio is a compiler suite for building C, C++, and Fortran applications for the Oracle Solaris and Linux operating systems. Oracle Developer Studio delivers tools optimized for the underlying operating system and hardware that support the development environment.

Oracle Developer Studio contains the following two major suites of tools:

- Compiler Suite – Helps you create applications in less time. This suite includes C and C++ compilers, a Fortran compiler, a debugger, and the Oracle Developer Performance Library.
- Analysis Suite – Helps you increase observability into your applications. This suite includes the Performance Analyzer, Code Analyzer and Thread Analyzer tools.

Oracle Developer Studio helps improve application performance and simplifies multicore development. Oracle Developer Studio is freely available for production use on Oracle Solaris and Linux operating systems. It is available both as an IPS package and a tar file. You can download Oracle Developer Studio 12.6 from the Download Options for Oracle Developer Studio (`https://www.oracle.com/technetwork/server-storage/developerstudio/downloads/index.html`).

Oracle Developer Studio also comes with an integrated development environment (IDE) that is built on the NetBeans platform. The IDE has a variety of features including an intelligent language-aware code editor, code completion, code folding, and highlighting. For more information, see Chapter 4, “Useful Tools and Tips for Developing Applications on Oracle Solaris”.

Other Tools for Application Development

Oracle Solaris provides the following tools for application development:

- Oracle JDeveloper – An integrated development environment that simplifies the development of Java-based SOA applications and user interfaces with support for the full development life cycle. For more information, see `https://www.oracle.com/java/`.
- NetBeans – An open-source environment, the NetBeans IDE supports the creation of enterprise, web, desktop, and mobile Java applications. All NetBeans IDE tools and features are fully integrated. For more information, see `https://netbeans.org/`.

For more information about other tools available for application development, see “Installing Software Useful for Application Development” in *Setting Up the Application Development Environment in Oracle Solaris 11*.

Checking Oracle Solaris 10 Applications for Oracle Solaris 11 Compatibility

The Preflight Applications Checker enables you to check whether your existing applications on Oracle Solaris 10 can run on Oracle Solaris 11. You can also use it to determine the Oracle Solaris 11 readiness of an application.

The tool checks the readiness of an application for Oracle Solaris 11 by performing an analysis in the following areas:

- Static analysis of the binary or C++ sources for the usage of deprecated, removed, unsupported, or unstable system calls that may not properly function on Oracle Solaris 11.
- Dynamic, DTrace based analysis of a running application for the usage of dynamic libraries that have been removed, relocated, or upgraded. For example, OpenSSL.
- Configuration files such as locales that have been removed or relocated.
- Commands that have been removed or relocated.

The Preflight Applications Checker generates a report in HTML format that lists all potential areas of consideration for a given application. For more information, see `https://www.oracle.com/technetwork/server-storage/solaris11/downloads/preflight-checker-tool-524493.html` and Appendix A, “Preflight Applications Checker”.

For information about transitioning to Oracle Solaris 11, see *Transitioning From Oracle Solaris 10 to Oracle Solaris 11.3*.

Data Migration Process

Data migration is necessary when an organization decides to use a new computing system or database management system that is incompatible with the current system. Typically, data migration is performed by a set of customized programs or scripts that automatically transfer the data.

Data migration can involve movement of data in file systems, files, applications, and databases. Some stored data might be in an encoded format or in a format that is incompatible with the receiving system. Note that both, Red Hat Enterprise Linux (RHEL) and Oracle Solaris use ASCII to store text data.

Migrating data involves the following activities:

- Migration of raw data including migration of application data, schema, tables, indexes, and constraints
- Migration of associated infrastructure such as stored procedures, database triggers, SQL queries, and functions

Endianness can be an issue when migrating data because the binary (raw) data stored in a file is usually not transferable between SPARC and x86/x64 platforms.

Applications storing data that is shared between platforms can handle endianness issues in one of the following two ways:

- Storing data in an application-defined, endian-neutral format by using text files and strings
- Choosing either the big-endian or little-endian convention and perform byte swapping by using enabling technology such as XDR

Migrating Databases

Moving a database from one platform to the other usually requires data transformation. During migration, if both platforms support a database from the same vendor, migration is simpler. For example, you can export the database running on Linux to a standardized file format, and then import it into a new database on Oracle Solaris. When the migration also involves a change in database vendors, more extensive data transformations might be required.

Oracle supports the migration of database objects and data from a variety of databases to Oracle Database. Oracle Database is optimally tuned in to get the most out of Oracle Solaris. You can use the Oracle SQL Developer tool to migrate from non-Oracle databases to Oracle Database. For more information, see “Oracle SQL Developer Tool for Migrating Databases” on page 17.

Oracle Solaris supports the following open-source databases:

- MySQL
- PostgreSQL
- SQLite
- Ingres
- Berkeley DB

The proprietary databases that are supported by the Oracle Solaris 11 OS are Oracle, DB2, Sybase, and Informix.

Considerations and Dependencies for Migrating a Database to Oracle Solaris

When you migrate a database to Oracle Solaris, you need to consider the following factors:

- Database support on the platform
- Database version compatibility
- Migration effort involved
- Licensing costs and migration of licenses

Installing Oracle Database

For a description of factors you need to consider before you install Oracle Database, see Oracle Database Pre-installation Tasks (https://docs.oracle.com/database/121/LADBI/pre_install.htm#LADBI222).

For information about installing Oracle Database on Oracle Solaris, see Oracle Database Online Documentation 12c Release 1 (https://docs.oracle.com/database/121/nav/portal_11.htm).

Oracle SQL Developer Tool for Migrating Databases

Oracle SQL Developer provides a graphical user interface tool that eases the database migration process by simplifying database development tasks. SQL Developer supports Oracle Database versions 10g, 11g, and 12c. You can browse, create and modify database objects, run SQL statements, edit and debug SQL, and access a list of predefined reports or create your own report. SQL Developer can connect to Oracle Database starting with version 10g and runs on the Windows, Linux, and Macintosh platforms. For more information, see SQL Developer: Migrating Third-Party Databases (https://docs.oracle.com/cd/E25259_01/appdev.31/e24285/migration.htm#RPTUG40000) and SQL Developer Documentation (https://docs.oracle.com/cd/E12151_01/index.htm).

♦ ♦ ♦ CHAPTER 3

Migrating From RHEL to Oracle Solaris

This chapter provides information about how to migrate from Red Hat Enterprise Linux (RHEL) to Oracle Solaris. It also discusses the differences between the technologies that are available in RHEL and Oracle Solaris.

Mapping Technology Between RHEL and Oracle Solaris 11

The following table highlights the key differences in technologies between RHEL and Oracle Solaris.

TABLE 1 Mapping Technology Between RHEL and Oracle Solaris 11

Task	Red Hat Enterprise Linux	Oracle Solaris 11
Packaging	RPM - A package manager that can be used to install, update, uninstall, and query for packages. Commands: `rpm`, `rpmdb`, and `rpmsign`	Image Packaging System (IPS) – Command line `pkg` and graphical Package Manager allows you to perform package operations such as install, update, uninstall, and query. Commands: `pkg`, `pkgsend`, `pkgrecv`, `pkgsign`, `pkgdiff`, `pkgfmt`, `pkgmogrify`, and `pkgrepo`
Services	`systemd` – The `systemd` system with service manager controls the startup and shutdown of system services.	Service Management Framework (SMF) – A framework that manages system and application services. SMF improves the availability of a system by ensuring that essential system and application services run continuously even during hardware or software failures. Commands: `svcadm`, `svccfg`, and `svcprop`
Network Virtualization	RHEL provides network virtualization support in KVM.	Network virtualization is administered at the datalink level. You can create VNICs that act like physical NICs. Virtual switches are automatically created to route the network traffic to the physical NIC device.

Task	Red Hat Enterprise Linux	Oracle Solaris 11
	Commands: `ip`, `brctl`, and `tunctl`	Commands: `dladm`, `flowadm`, `dlstat`, `flowstat`, `evsadm`, and `evsstat`
Virtualization	KVM – A full virtualization solution on Red Hat Enterprise Linux. Commands:`virsh`, `virt-clone`, `virt-convert`, `virt-image`, `virt-install`, `virt-viewer`, `virt-what`, and `virt-xml-validate`	Oracle Solaris Zones – Provide native low-overhead OS virtualization with high application isolation and resource management. Commands: `zoneadm`, `zonecfg`, `zonestat`, `zonename`, and `zone2pvhck` Oracle Solaris also supports Oracle VM Server for SPARC platform and Oracle Solaris Kernel Zones.
File Systems	Ext4, XFS, LVM, btrfs, and UFS	Oracle Solaris ZFS – Default file system on Oracle Solaris 11. Maximum file size of 16 EB, maximum volume size of 16 EB. Oracle Solaris ZFS has integrated data services such as snapshot and cloning, deduplication, encryption, compression, shadow migration, and RAID. Commands: `zfs` and `zpool`
Encryption	Linux Unified Key Setup (LUKS) – RHEL supports LUKS for file system encryption. LUKS protects data in a partition that has been encrypted only when the system has been turned off. Command: `cryptsetup`	ZFS – Supports full encryption of datasets during creation. Command: `zfs` Oracle Solaris Cryptographic Framework – Provides a common store of algorithms and PKCS #11 libraries to handle cryptographic requirements. Commands: `cryptoadm` and `pktool`
Monitoring	SystemTap – Provides dynamic instrumentation of RHEL. Commands: `stap`, `staprun`, `stap-report`, `stapsh`, `stap-merge`, and `stap-prep`	DTrace – A comprehensive dynamic tracing facility that can be used by administrators and developers on live production systems to examine the behavior of both user programs and the operating system itself. Command: `dtrace` Oracle Enterprise Manager Ops Center 12c, included with all Oracle Premier Support agreements, provides extensive monitoring at a greater scale including both Oracle Solaris and Linux systems. For more information, see `https://`

Task	Red Hat Enterprise Linux	Oracle Solaris 11
		www.oracle.com/technetwork/oem/ops-center/index.html.

Threading Models in Oracle Solaris and RHEL

On RHEL, two 1:1 threading implementations are provided by the GNU C library (`glibc`). Each thread maps to a kernel scheduling entity.

The two threading implementations are as follows:

- LinuxThreads – The original Pthreads implementation. Since `glibc 2.4`, this implementation is no longer supported.
- Native POSIX Threads Library (NPTL) – The latest Pthreads implementation.

Oracle Solaris uses the 1:1 thread model in preference to the MxN implementation. Simplifying the underlying thread implementation for existing applications improves performance and stability without requiring recompilation.

Oracle Solaris supports both Pthreads and Oracle Solaris threads. You can use the Pthread model for new application development and porting efforts. Over 100 standards POSIX functions are available through the Pthreads API. For more information, see the `pthreads(5)` man page and *Multithreaded Programming Guide*.

To maximize performance, you can modify applications to execute many tasks simultaneously. Oracle Solaris provides support for concurrent processing such as multiple threads, shared memory, and asynchronous I/O. Multithreaded application programs aim to improve parallelism, and can be written without regard to the number of CPUs configured on the target system.

For more information about the threading models, see Red Hat Enterprise Linux to Oracle Solaris Porting Guide.

Migrating Services From RHEL to Oracle Solaris

You can migrate services from RHEL to Oracle Solaris Service Management Facility (SMF) because SMF provides compatibility with legacy services. Legacy services include `/etc/rc*.d`, `/etc/init.d`, and `/etc/inittab` scripts, which are usually used in RHEL environments. Legacy services can continue to work as they did previously, and you can observe these services

with SMF. However, to exploit all the advantages of SMF, such as self-healing capabilities or service restart, you need to convert the scripts to SMF manifests. For more information, see "Service Manifests in Oracle Solaris" on page 22.

SMF on Oracle Solaris 11 enables you to perform the following tasks through a single framework:

- Observe and manage system-wide services
- Provide consistent configuration handling
- Automatically restart failed services in the appropriate order of dependency
- Identify failed services
- Securely delegate administrative tasks to non-root users
- Preserve compatibility with legacy services
- Automatically configure system administration jobs such as backup and restore

For more information about SMF, see *Managing System Services in Oracle Solaris 11.3*.

To increase system availability, the Fault Management Architecture (FMA) in Oracle Solaris helps you to detect system problems. For more information, see "Fault Management Overview" in *Managing Faults, Defects, and Alerts in Oracle Solaris 11.3*.

Service Manifests in Oracle Solaris

Information regarding services, service properties, property groups, and instances is stored in a service manifest file, `/lib/svc/manifest`, which is transferred to the SMF repository. SMF uses manifest information when managing services and when determining the root causes of service failures. The service manifest also describes the conditions under which failed services might be automatically restarted. A separate service manifest is required per service or application. Oracle Solaris provides some service manifests by default. Optionally, you can customize these manifests or write your own for other services. For more information, see *Managing System Services in Oracle Solaris 11.3*.

Best Practices for Moving Applications to SMF and FMA

The following best practices can facilitate the migration of applications to the SMF and FMA framework:

- Eliminate custom scripts that analyze application health and restart applications. SMF provides a simple way to encapsulate and standardize the methods used to start, stop, and restart applications.
- Make applications SMF-aware early in the porting and testing process. Identify fault states and create a fault tree. Review error messages that are encountered and determine whether they can become FMA events.
- Convert `.rc` and custom scripts to SMF profiles. Look for instances of start, stop, and check status methods.

Packaging in Oracle Solaris

Oracle Solaris 11 uses the Image Packaging System (IPS) for package management. IPS simplifies the process of managing patches and updates to reduce the risk of operating system maintenance issues. It is a comprehensive delivery framework that spans the complete software life cycle, addressing software installation, updates, system upgrades, and the removal of software packages. IPS relies on network accessible or locally available software repositories as a delivery mechanism. For more information, see *Copying and Creating Package Repositories in Oracle Solaris 11.3*.

Each IPS package is represented by a Fault Management Resource Identifier (FMRI), that provides information about the package. For example, the FMRI `pkg://solaris/system/library@0.5.11,5.11-0.151.0.1:20101105T004750Z` consists of the following sequence of information:

- Scheme – `pkg`
- Publisher – `Oracle Solaris`
- Category – `system`
- Package name – `library`
- Version string, which consists of four components :
  - Component version – `5.11`
  - Build version – `5.11`
  - Branch version – `151.0.1`
- Timestamp in ISO-8601 basic format: `20101105T004750Z`

For more information about packaging and delivering, see *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.3*.

♦ ♦ ♦ CHAPTER 4

Useful Tools and Tips for Developing Applications on Oracle Solaris

This chapter discusses various tools and tips available in Oracle Solaris that enables you to fine-tune your application. You can take advantage of the features supported by these tools to develop applications and run them optimally on Oracle Solaris.

Optimizing Applications by Using Oracle Developer Studio

The tools provided by Oracle Developer Studio enable you to leverage the processing capabilities of the latest CPU architectures such as the Oracle SPARC T-Series and M-Series, Intel Xeon, and AMD Opteron processors. The tools enable easier creation of parallel and concurrent software applications for these platforms. For more information about SPARC M-Series and T-Series servers see https://docs.oracle.com/en/servers/sparc.html and for other Oracle servers, see x86 Servers.

If the migrated application does not have strict requirements for adherence to standards, you can obtain additional performance gains by using optimization flags to build higher-performing binaries. For example, the `-fast` macro in Oracle Developer Studio is the easiest way to generate an optimized binary for specific target hardware.

Note - The `-fast` macro implies the `-fns` option.

This section provides information about optimizing applications by using Oracle Developer Studio.

Oracle Developer Studio Tools

To generate high-performing binaries and isolate code issues that are difficult to detect, you can use the following Oracle Developer Studio tools:

- Performance Analyzer – Analyzes application performance, identifies performance hotspots, and determines the parts of a program that need to be improved.
- Thread Analyzer – Detects code issues in multithreaded programs. For example, you can detect data races and deadlock conditions.
- Discover – Detects programming errors related to the allocation and use of program memory at runtime. You can detect the following types of programming errors:
  - Reading from and writing to unallocated memory
  - Freeing the wrong memory blocks
  - Using free memory
  - Accessing memory beyond allocated array bounds
  - Memory leaks
  - Accessing uninitialized memory
- Uncover – Measures the code coverage of user applications. The coverage information reported by this tool could be at the level of a function, statement, basic block, or instruction By default, the high-performance `libumem` library is bundled with Oracle Solaris 11. The `libumem` library, along with the `dbx` debugging tool, finds memory leaks and buffer overruns that are difficult to detect.

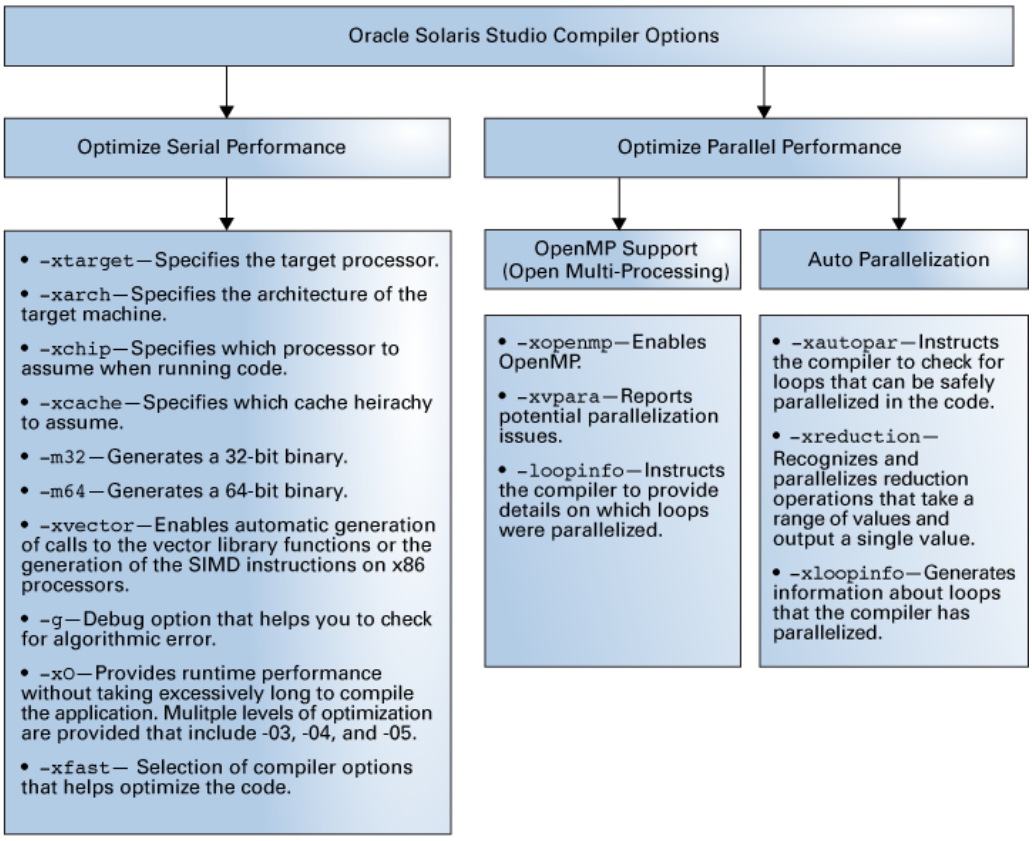
For more information, see the Oracle Developer Studio 12.6 Documentation Library.

Optimizing the Serial and Parallel Performance of Applications

Application performance is viewed in the context of hardware, operating systems, and serial and parallel environments. For applications running in a serial environment, you can optimize the performance by using appropriate compiler flags that help the applications use hardware resources efficiently. For applications running in a parallel environment, you can use the compiler options to take advantage of multiple cores and multiple threads of execution to optimize performance of the applications. You can use auto-parallelization and OpenMp compiler flags to build and run parallel versions of an application.

The following figure shows the Oracle Developer Studio compiler options that are used for optimizing serial and parallel performance of applications.

FIGURE 1 Oracle Developer Studio Compiler Options



Choosing Compiler Optimization Options

Optimization flags affect the following three important characteristics:

- The compilation time
- The runtime of the compiled application
- The amount of debug activity that is possible with the final binary

In general, the higher the level of optimization, the faster the application runs and the longer it takes to compile, but the less debug information is available. The impact of optimization levels varies from application to application.

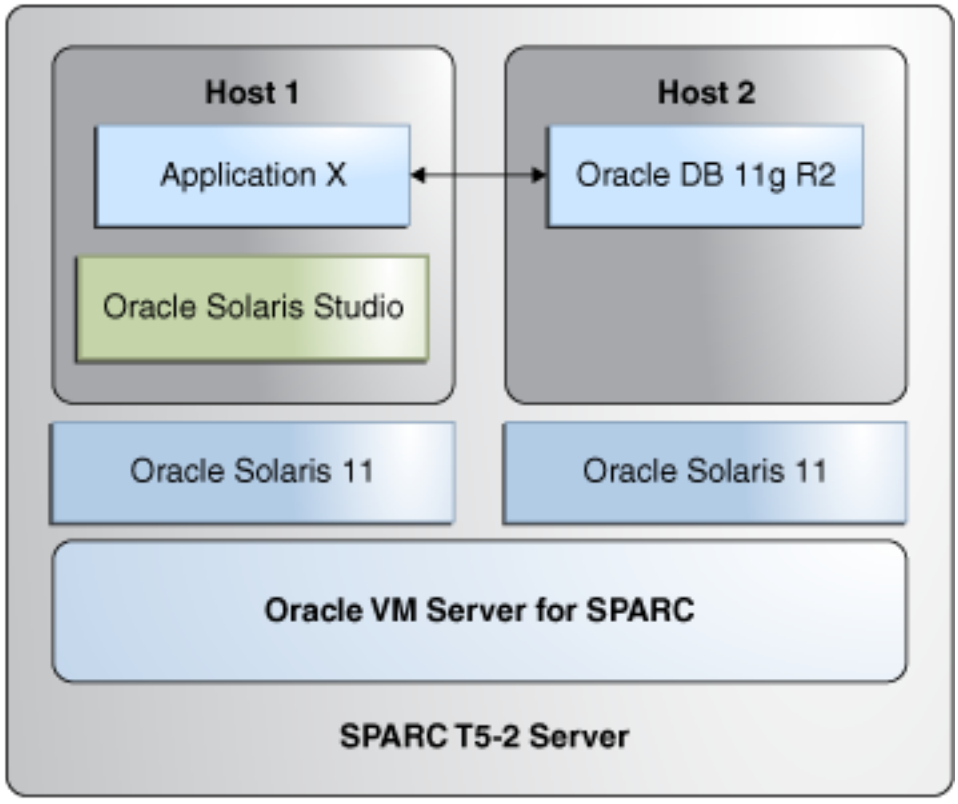
Use Case: Optimizing the Performance of Applications

This use case presents a scenario in which you optimize the performance of an application by using Oracle Developer Studio. The following activities are performed in this use case:

- Using performance analyzer tool to understand the resource utilization of the application
- Optimizing the application for serial and parallel performance by using the compiler options
- Optimizing the binary, based on the runtime behavior of the application by using feedback profiling

This use case assumes the architecture shown in the following figure.

FIGURE 2 Performance Optimization Use Case Architecture



The example architecture consists of the following setup:

- SPARC T5-2 server with 32 cores and 256 GB RAM.
- The server is partitioned by using the Oracle VM Server for SPARC 3.1 platform. A separate partition with 16 cores and 128 GB of RAM is used for deploying applications. The primary domain with 16 cores and 128 of GB RAM is used to host the Oracle Database 11gR2.
- Oracle Solaris 11 is configured on both the application server and the database server.
- Oracle Developer Studio 12.6 is installed on the application server.
- Application X is running on the application server.

Using Performance Analyzer

The Oracle Developer Studio Performance Analyzer enables you to study the behavior of the application and determine the parts of the application that need to be improved. The `collect` command is used for the collection of data and the `analyzer` command to invoke the performance analyzer.

You can analyze the following key areas:

- Variable memory allocation of larger sizes
- Execution time statistics
- Time spent by the application in various functions
- Hardware counters to find issues such as cache misses

Using Compiler Options for Optimizing the Application

You can use the compiler options that are provided by the Oracle Developer Studio compiler to optimize the application. The compiler options that are used in this use case are described in the following table.

TABLE 2 Compiler Options

Compiler Options	Function
`-DSOLARIS`	Assigns the macro symbol Solaris to the preprocessor.
`-fast`	Provides maximum runtime performance. This option provides the following individual compilation optimizations:
	`-fns` – Enables the nonstandard floating-point mode in SPARC platforms.
	`-fsimple=2` – Selects floating-point optimization preferences.
	`-xarch` – Specifies the target instruction set architecture.
	`-xbuiltin=%all` – Enables better optimization of standard library calls.
	`-xcache` – Defines cache properties that are used by the optimizer.
	`-xchip` – Specifies the target processor that is used by the optimizer.
	`-xlibmil` – Inlines the selected `libm` library routines for optimization.
	`-xlibmopt` – Uses library of optimized math routines.
	`-xmemalign` – Controls the assumptions that the compiler makes about the alignment of data.
	`-xO5` – Generates the highest level of optimization.
`-m64`	Specifies 64-bit memory model for the compiled binary object.
`-xtarget=T5`	Specifies the target platform for instruction set and optimization.

Compiler Options	Function
`-xautopar`	Enables automatic parallelization for multiple processors.
`-xdepend`	Analyzes loops for inter-iteration data dependencies and does loop restructuring, including loop interchange, loop fusion, scalar replacement, and elimination of “dead array” assignments.
`-xprefetch=auto`	Enables automatic generation of the `prefetch` instructions.
`-xcode=pic32`	Generates position-independent code (large model), which might not be as fast as `pic13`, but has full range. Equivalent to `-KPIC`. Permits references to at most 2**30 unique external symbols on 32-bit architectures; 2**29 on 64-bit.
`-xipo=2`	Performs inter-procedural aliasing analysis as well as optimizations of memory allocation and layout to improve cache performance.
`-xlinkopt=2`	Instructs the compiler to perform link-time optimization on the resulting executable or dynamic library in addition to any optimizations in the object files. Performs additional data flow analysis, including dead-code elimination and address computation simplification, at link time.

Using Oracle Developer Studio Feedback Profiling

Oracle Developer Studio feedback profiling enables you to optimize the binaries of the application. In feedback profiling, you build the application twice, once to collect the profile data and again to make use of the profile to generate an optimal code. This use case uses the following procedure for feedback profiling.

1. Sets the environment variable `SUN_PROFDATA_DIR` to provide the location of the profile directory that will be generated.

   ```
   # setenv SUN_PROFDATA_DIR /tmp/consolidate
   ```

   The profile directory is stored in the `/tmp/consolidate` folder.
2. Sets the environment variable `SUN_PROFDATA` to record the profile data of the product in a single file.

   ```
   # setenv SUN_PROFDATA singlefeedbin.profile
   ```

   The profile data is stored in the `singlefeedbin.profile` file.
3. Compiles the binaries by using the `-xprofile` option.

   ```
   -xprofile={collect,tcov}:xxx.profile
   ```

   The `-collect` option helps in collection of data about the program and the `-tcov` option provides the code coverage after the execution of the program.
4. Runs the application with representative workloads that resembles the real-time environment.

The information about the behavior of the program during these runs is saved in the `.profile` directory that is created in the step 2.

5. Recompiles the binaries by using the `profile` data.

```
-xprofile=use:/path/to/xxx.profile
```

The `-use` option performs the following optimizations based on the profile data:

- Code layout
- Register allocation
- Loop transformations
- Branch optimizations
- Block straightening
- Switch case code generation
- Global instruction scheduling
- Delay slot scheduling
- Branch prediction

Addressing Multithreaded Application Issues

This section describes how to address multithreaded application issues in Oracle Solaris.

Using Thread Analyzer

The Oracle Developer Studio Thread Analyzer is an advanced tool for application optimization, and ensures multithreaded application correctness. Specifically, the Thread Analyzer can detect, analyze, and debug the special situations that can arise in multithreaded applications. For more information, see *Oracle Developer Studio 12.6: Thread Analyzer User's Guide*.

Detecting Race Conditions and Deadlocks

To instrument the source code to detect race conditions and deadlock issues, you can compile the source code with a special flag, executed under the control of the `collect -r` command, and then load the program into the Thread Analyzer.

1. Applications are first compiled with the `-xinstrument=datarace` compiler flag. To help ensure that the line numbers and call-stacks information are returned correctly, set the `-g` flag and do not specify an optimization level.
2. The resulting application code is then executed after the `collect -r` command is issued, which collects key runtime information. Use the `collect -all` option to run the program and create a data race detection and deadlock detection experiment during the execution of the process.

   ```
   $ collect -r race app params
   $ collect -r deadlock app params
   ```
3. The results of the experiment are loaded into the Thread Analyzer to identify data race and deadlock conditions.

Debugging Applications by Using DTrace

DTrace is a comprehensive dynamic tracing framework used for troubleshooting kernel and application problems on production systems in real time. It helps track down performance problems across many layers of software, and locates the cause of the unexpected behavior. DTrace is a powerful dynamic tracing tool that can trace any part of the system.

The salient features of DTrace are as follows:

- Enables dynamic modification of the system to record arbitrary data
- Can be used on any application without having to recompile and restart the application
- Uses very little additional system calls when a system is activated
- Allows tracing of both the kernel and user-level programs
- Functions with minimal overhead when tracing is enabled and zero overhead when tracing is not being performed
- Does not stop the program's execution after every system call and therefore can be used on production systems without fear of a crash
- Can be used on production systems to find performance bottlenecks
- Enables maximum resource utilization and application performance, as well as precise quantification of resource requirements

For more information about DTrace, see *Oracle Solaris 11.3 DTrace (Dynamic Tracing) Guide*.

Debugging Applications by Using dbx

The dbx debugger component of Oracle Developer Studio helps find bugs in code at the command line, through the IDE, or through an independent graphical interface (dbxtool). You can also use dbx to inspect the state of a stopped program. dbx gives you complete control of the dynamic execution of a program including collecting performance and memory usage data, monitoring memory access, and detecting memory leaks. Using dbx, you can also navigate between threads, suspend threads, and display the stack and locks.

dbx can be used to debug an application written in C, C++, Java, or Fortran. However, there are certain limitations while debugging Java code.

File Systems in Oracle Solaris

The file systems that are supported by Oracle Solaris 11 are as follows:

- Disk-based file systems: HSFS, PCFS, UDFS, UFS, and ZFS
- Network-based file systems: NFS and SMB
- Virtual file systems: CTFS, FIFOFS, MNTFS, NAMEFS, OBJFS, SHAREFS, SPECFS, and SWAPFS
- Temporary file system (TMPFS)
- Loopback file system (LOFS)
- Process file system (PROCFS)

UFS is a legacy file system, but it is not supported as a bootable root file system. ZFS is the default root file system. For more information about ZFS file systems in Oracle Solaris, see the Data Management web page and *Transitioning From Oracle Solaris 10 to Oracle Solaris 11.3*.

Security and Privileges in Oracle Solaris

Oracle Solaris provides a network-wide security system that controls the way users access files, and protects system databases and system resources. It combines multiple security technologies such as networking, cryptographic capabilities, and trusted extensions to manage user rights.

Some of the highlights of security-related features in Oracle Solaris are:

Compliance checking and reporting	You can administer security compliance tests by using the `compliance` command. You can assess and report the compliance of an Oracle Solaris system with security standards, also called security benchmarks and security policies. For more information, see *Oracle Solaris 11.3 Security Compliance Guide*.
Verified boot	An anti-malware and integrity feature that reduces the risk of introducing malicious or accidentally modified critical boot and kernel components. This feature checks the cryptographic signatures of the firmware, boot system, and kernel and kernel modules. Verified boot applies to the SPARC T5, SPARC M5, and SPARC M6 platforms. For more information, see "Using Verified Boot" in *Securing Systems and Attached Devices in Oracle Solaris 11.3*.
Support for IKEv2	Provides automatic Security Association (SA) and key management between peer systems.
RBAC Time-Based and Location-Based Access	You can qualify user attributes by location. A new qualifier option for the `usermod` and `rolemod` commands can indicate the system or netgroup where user attributes apply. A new time-based policy for access to PAM services can be specified by using the new `access_times` keyword of the `useradd` command. For more information see the `usermod`(1M), `rolemod`(1M), and `useradd`(1M) man pages.

Oracle Solaris Trusted Extensions

Oracle Solaris Trusted Extensions is a set of advanced security features that allow you to label the data and applications according to the sensitivity level. It features the access control model that includes RBAC, Mandatory Access Control Labeling, Auditing, and Device Allocation. It is an optional layer of secure label technology in Oracle Solaris that allows data security policies to be separated from data ownership.

Trusted Extensions provides APIs to develop application will allow you to access and handle labels. For more information about Trusted Extensions APIs, see *Trusted Extensions User's Guide*.

Authentication Services in Oracle Solaris

Authentication is a mechanism to validate whether a user or service matches the predefined criteria.

The main features of the Oracle Solaris authentication services are as follows:

- Secure RPC – Protects remote procedures using the Diffie-Hellman authentication mechanism.
- Pluggable Authentication Module – Enables you to add new authentication methods and modify the authentication policies by installing PAM modules into the Oracle Solaris OS.
- Simple Authentication and Security Layer – Provides authentication and security services to network protocols.
- Secure Shell – Provides secure data communication and remote command-line login using cryptographic network protocol.

Pluggable Authentication Module

Pluggable Authentication Module (PAM) is a set of pluggable objects that enables system administrators to add new authentication services without changing system services. You can use it to modify user authentication, account, session, and password management functions in Oracle Solaris. Login, `ssh`, and other system entry services use the PAM framework to ensure that all login sessions are secure. Flexibility to modify the configuration files is the main feature that benefits the users.

For more information about PAM modules, see Chapter 3, "Writing PAM Applications and Services" in *Developer's Guide to Oracle Solaris 11.3 Security*.

For more information about the differences between the RHEL and Oracle Solaris implementations of PAM, see the section *Pluggable Authentication Module* in Red Hat Enterprise Linux to Oracle Solaris Porting Guide.

Oracle Solaris Cryptographic Framework

The Oracle Solaris Cryptographic Framework provides a set of cryptographic services for kernel-level and user-level consumers. Oracle Solaris provides network security based on standard industry interfaces such as PAM, GSS-API, SASL, and PKCS #11. The Cryptographic Framework is a backbone of cryptographic services in Oracle Solaris. The framework provides standard PKCS#11 interfaces to accommodate consumers and providers of cryptographic services.

The framework has two parts:

- User cryptographic framework for user-level applications
- Kernel cryptographic framework for kernel-level modules

The main elements of the Oracle Solaris Cryptographic Framework are as follows:

- `libpkcs11.so` library – The framework provides access through the RSA Security Inc. PKCS#11 Cryptographic Token Interface (Cryptoki). Applications must link to the `libpkcs11.so` library.
- `pkcs11_softtoken.so` shared object – Contains user-level cryptographic mechanisms provided by Oracle.
- `pkcs11_kernel.so` shared object – Used to access kernel-level cryptographic mechanisms. It offers a PKCS#11 user interface for cryptographic services that are plugged into the kernel's service provider interface.
- Pluggable interface – The pluggable interface is the service provider interface (SPI) for PKCS #11 cryptographic services that are provided by Oracle and third-party developers. Providers are user-level libraries that are implemented through encryption services available through the hardware or software.
- Scheduler and Load Balancer – Enables efficient load balancing, and dispatching cryptographic requests.
- Kernel Programmer Interface – Provides kernel-level consumers with access to cryptographic services.
- Service Provider Interface – Used by providers of kernel-level cryptographic services that are implemented in the hardware and the software.
- Hardware and software cryptographic providers – Kernel-level cryptographic services that use software algorithms, hardware accelerator boards, or on-chip cryptographic capabilities.
- Kernel cryptographic framework daemon – Private daemon responsible for managing system resources for cryptographic operations. The daemon also verifies cryptographic providers.
- Module Verification Library – Private library used to verify the integrity and authenticity of all binaries that the cryptographic framework imports.
- `elfsign` – Utility offered to third-party providers of cryptographic services to request certificates from Oracle.
- `cryptoadm` – User-level command for managing cryptographic services, such as disabling and enabling cryptographic mechanisms according to security policies.

Four types of applications can plug into the Oracle Solaris Cryptographic Framework:

- User-level consumers
- User-level providers
- Kernel-level consumers
- Kernel-level providers

The Oracle Solaris Key Management Framework provides tools and programming interfaces for managing public key infrastructure (PKI) objects.

The on-board cryptography in Oracle Solaris servers like UltraSPARC T1, UltraSPARC T2, and UltraSPARC T2+ on-chip cryptographic acceleration eliminates the need for additional coprocessor cards, or power-consuming add-on components. For more information about different function calls in RHEL and Oracle Solaris, see Red Hat Enterprise Linux to Oracle Solaris Porting Guide.

For more information about the Oracle Solaris Cryptographic Framework, see Chapter 9, "Writing User-Level Cryptographic Applications" in *Developer's Guide to Oracle Solaris 11.3 Security*.

High Availability in Oracle Solaris

High availability (HA) ensures that a system, application, or a database is available, continuously accessible, and running for the desired period of time without any loss of service.

Oracle Solaris offers application and database availability. High availability for applications is provided by Oracle Solaris Cluster, which provides load balancing, fault detection, and disaster recovery to keep the application or the system highly available. Oracle Solaris Cluster provides an extensive high-availability framework. Oracle Solaris provides a full range of single and multisystem high-availability in traditional and virtualized environments.

Oracle Solaris Cluster software's high-availability framework detects a node failure and migrates the resources to another node in the network that can take up the action.

Oracle Solaris Cluster can also track, maintain, and monitor zones. It also offers a failover zone feature to protect a zone when it fails. The failover zone is monitored by HA agents for zones. HA Oracle Agent software controls activities in Oracle Database activities on Oracle Solaris Cluster nodes. The agent performs fault checking by starting, stopping, and probing the nodes. If an HA agent is configured on a database that has no active sessions, HA Oracle Agent will open a test transaction. The return error codes from HA Oracle Agent will be validated against a special action file on location.

Oracle Application Server provides local and enterprise high availability using the technologies like process death detection and automatic restart, clustering, server load balancing, rolling and patching, and backup and recovery. The Oracle Clusterware software provides infrastructure to manage any application.

Oracle Solaris Cluster includes an Agent Builder tool for application high-availability that automates the creation of data services. Developers supply Agent Builder with information about the application and data service to be created, such as whether a scalable or failover agent is desired, whether the service is network-aware, and the commands to use to start and stop the application. Agent Builder generates the data service, including source and executable code

(C or Korn shell), a customized Resource Type Registration (RTR) file, and an Oracle Solaris package for distribution.

For more information about application server high availability, see https://docs.oracle.com/cd/E12839_01/core.1111/e10106.pdf.

Oracle Database provides high availability by minimizing both planned and unplanned downtime. Oracle Database achieves high-availability using Oracle Real Application Clusters (RAC) and Oracle Clusterware software. Together, they bind the servers to operate as one server. Some of the benefits of Oracle RAC and Oracle Clusterware software are:

- Ability to tolerate and quickly recover from computer and instance failures
- Ability to apply Oracle Clusterware upgrades, patch sets, and interim patches in a rolling fashion
- Fast, automatic detection of connection failures and removal of terminated connections for any Java application using Oracle Universal Connection Pool (UCP), Fast Connection Failover, and FAN events
- Comprehensive manageability integrating database and cluster features
- Automatically restarts failed Oracle processes
- Automatically manages and fails over Oracle Virtual IP (VIP) on another node in the cluster on node failures

For more information about Oracle Database high availability, see https://www.oracle.com/database/high-availability/index.html.

For more information about Oracle Solaris Cluster, see Oracle Solaris Cluster Product Documentation.

Network Virtualization in Oracle Solaris

Network virtualization is the process of combining hardware network resources and software network resources into a single administrative unit. This single administrative unit is known as a virtual network. In Oracle Solaris, virtual network interface cards (VNICs) and etherstubs are the basic components of a virtual network. You can create VNICs over physical datalinks. The configured VNICs behave like physical NICs. An etherstub is a pseudo Ethernet NIC that is configured at the datalink layer (L2) of the Oracle Solaris network stack.

Starting with the Oracle Solaris 11.2 release, you can use the Oracle Solaris Elastic Virtual Switch (EVS) feature that enables you to create and administer a virtual switch that spans one or more compute nodes. For more information about network virtualization and EVS, see *Managing Network Virtualization and Network Resources in Oracle Solaris 11.3*.

Oracle Solaris Remote Lab

Oracle Solaris Remote Lab (OSRL) is a cloud-based self-service lab that enables Oracle partners with access to a remotely accessible environment to validate and certify their applications on Oracle Solaris 11.

Oracle partners can allocate up to five virtual machines (VMs) with a combination of SPARC and x86 processors. Each partner's VMs share an NFS mounted file system in a private network that segregates all network traffic.

♦ ♦ ♦ APPENDIX A

Preflight Applications Checker

The Oracle Solaris Preflight Applications Checker "Preflight Checker" enables you to identify potential issues when migrating Oracle Solaris 10 applications to Oracle Solaris 11. The tool runs a combination of source code, static binary, and runtime analysis modules on Oracle Solaris 10 to test existing applications and suggests the necessary code changes for the application to run successfully on Oracle Solaris 11.

Note - The Oracle Solaris Preflight Applications Checker should be used only on development or test systems, never on production systems.

About the Preflight Checker

The Oracle Solaris Preflight Applications Checker looks for potential issues and suggests better ways to implement application on an Oracle Solaris 11 system. The Preflight Applications Checker also includes the Oracle Solaris Preflight Kernel Checker, which helps you to find potential trouble spots in kernel modules. It includes the Source Code Analyzer and Binary Analyzer modules.

The primary purpose of these tools is to help developers test the readiness of an application before migration.

For more information, see the Oracle Solaris Guarantee Program.

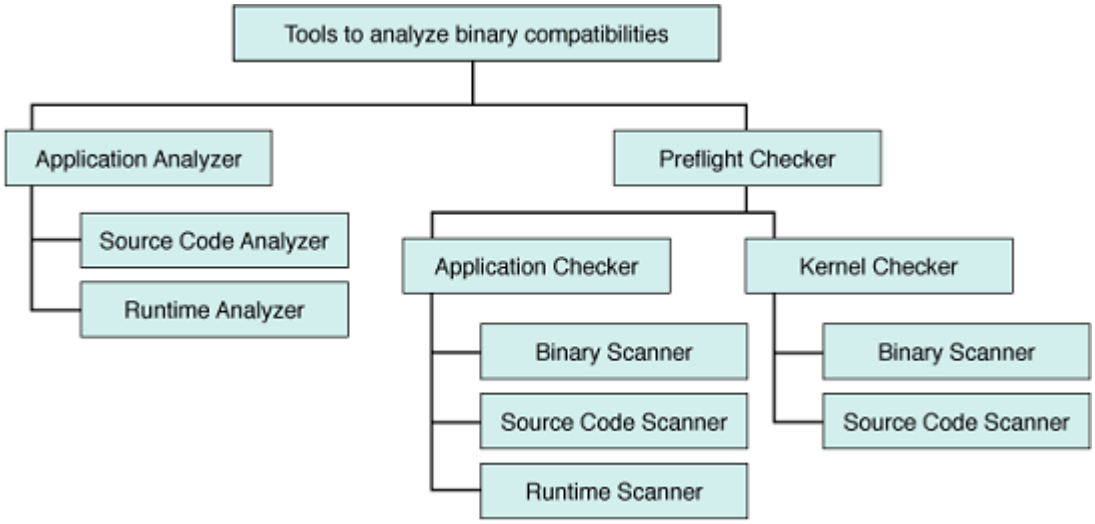
Preflight Checker Modules

The Preflight Applications Checker contains the following three main modules:

- Binary Analyzer
- Source Code Analyzer

- Runtime Analyzer

The following image shows the modules that are available to analyze binary compatibilities at a high level.



Binary Analyzer Module

The Binary Analyzer module, also called the ELF analyzer, extracts all the bindings and associated symbol information from the application binaries. If running the application requires setting the `LD_LIBRARY_PATH` environment variable or if the system's default runtime linking environment is changed by using `crle` or `LD_CONFIG`, you can provide a value for `LD_LIBRARY_PATH`.

The Preflight Checker uses the `LD_LIBRARY_PATH` value to determine the libraries being used by the application during runtime. This information helps to match the symbol bindings to the Oracle Solaris system shared libraries. While doing the binary analysis, if the application does not need the `LD_LIBRARY_PATH` variable to be set, the value must be blank. The `lib` path `/usr/lib/64` (`/usr/lib` or `/lib/64`) is automatically used by the Binary Analyzer module as with `crle`.

The Preflight Checker has a built-in database populated with potential issues when migrating from Oracle Solaris 10 to Oracle Solaris 11. This database is accessed for every system symbol used in the application. Based on the output from the database lookup, the tool reports appropriate potential issues along with potential workarounds or suggestions to overcome the issues.

Source Code Analyzer Module

The Source Code Analyzer module checks for the use of removed, renamed, modified, or deprecated library functions in Oracle Solaris. This module also checks for issues in C and C++ source code, and detects potential issues in the shell scripts.

Note - The default system shell `/bin/sh` and korn shell `/bin/ksh` are replaced by the implementation of `ksh93` in Oracle Solaris 11.

The shell scanner module helps to locate potential issues and report warning messages. In addition, this module also reports the use of files that are not available on Oracle Solaris 11.

The shell scanner module tries to resolve shell variables during scan time by looking into shell scripts available inside the target application directory. To get better results, you can set the environment variables outside the target scan directory and populate the `init_script` field with a comma-separated list of scripts through which the application environment is set. For example:

```
init_script = /export/home/user/.profile,/export/home/user/.bashrc,/export/home/user/
setenv.sh
```

Note - If the runtime environment information is not available during the static analysis of shell scripts, the shell scanner module might generate false positive results.

Runtime Analyzer Module

The Runtime Analyzer module can run on an existing Oracle Solaris 10 test environment and report the potential issues. This module of the Preflight Checker uses DTrace to look into process details. This module might also invoke other system tools available on Oracle Solaris 10.

You can configure the Runtime Analyzer module to run with any of the following scopes:

- System wide
- All processes running inside a specific zone
- All processes with a specific UID
- All running processes for a given executable name
- Specific PIDs
- A user-specified custom command

Note - The Runtime Analyzer module analyzes only user-initiated processes and ignores system processes.

The list of processes that are filtered out is maintained in the *tool-base-dir*`/conf/SystemProcs.conf` editable configuration file.

The Runtime Analyzer module can report known issues in the following cases:

- Invocation of removed, renamed, or relocated libraries, scripts, commands and devices, or other system-provided files.
- Use or invocation of deprecated private or obsolete functions, APIs, and symbols.
- `dlopen()` of removed, renamed, or moved libraries.
- Use or invocation of private interfaces.
- Use of outdated and removed versions of utilities or packages, for example, hard-coded paths to files that have been removed.
- Use of files or libraries that are upgraded or replaced in a newer release of the Oracle Solaris OS.

Considerations When Using the Preflight Checker

If you use the `dlopen()` or `mmap()` functions, the Static Binary Analyzer and the Source Code Analyzer cannot access the shared object locations because the application is accessing a shared object that does not have its Oracle Solaris dependencies recorded. However, the Runtime Analyzer can access libraries loaded with `dlopen()` function and files loaded into memory by using `mmap()`. For more information, see the `dlopen`(3C) and `mmap`(9E) man pages.

Other objects that Binary Analyzer and Source Code Analyzer cannot examine are as follows:

- Executable files that do not have read/execute permission set
- Non-ELF file executables

Note the following guidelines when using the Preflight Checker:

- You must provide correct environment details for the application while running the Preflight Checker. Differences in the environment could result in false positives for potential binary incompatibility or unresolved references to interfaces in the Oracle Solaris library. For example, if the `LD_LIBRARY_PATH` is not set correctly, the Preflight Checker reports additional unbound symbols.
- To help ensure accurate Preflight Checker reporting, record dynamic library dependencies at compilation time by using the `-z defs` link option of Oracle Developer Studio 12.6. This option makes the object self-contained and better defined. It forces a fatal error if

any undefined symbol remains at the end of the link. This option is the default when an executable is built.

- The Runtime Analyzer module uses DTrace to look into process details, and might invoke other system tools available on Oracle Solaris 10. To limit the overhead of the Runtime Analyzer on overall system performance and responsiveness, restrict its scope to the minimum possible level. For example, avoid doing a system-wide runtime analysis if you already know specific PIDs of the targeted application.

Understanding Preflight Checker Reports

Oracle Solaris Preflight Applications Checker reports list the analyzed files (for binary and source code analysis) and processes (for runtime analysis) that are targeted during the corresponding phases of the scan.

A summary appears at the beginning of the report indicating the number of binaries examined and the number of potential binary stability problems that were found. The summary report can also be found under *tool-install-dir*/`reports`. By default, the report is created in HTML format.

Reports also include suggestions for actions to be taken to resolve potential issues. Common issues reported by Preflight Checker after running a scan include the following:

- Private symbol usage – If the symbol has not changed in the current release then the application can still run. However, future use of the symbol is not guaranteed and could cause problems. Eliminate the use of such private symbols.
- Static linking – If an application is built by statically linking with `libc.a`, any kernel interface that it references is extracted from the `libc.a` archive and becomes a part of the application. This application runs only on a kernel that is in sync with the kernel interfaces used. Because you must link your application with shared versions of system-provided libraries, if the kernel interfaces change, the application might break.
- Demoted or deprecated symbol – Eliminate the use of deprecated symbols.
- No Bindings Found – Check environment variables such as `LD_LIBRARY_PATH` to ensure that they are correctly set so that the binary objects can find all of the libraries they depend on. For more information about how to cope with the possible problems resulting in unbound symbols, see the man pages of the corresponding APIs.
- Obsolete library usage – Eliminate use of these symbols as soon as practical.

Defaults Used by the Preflight Checker

If you do not provide custom information during the installation phase for the Preflight Checker or while starting the analysis, the tool uses the following default values:

- LD_LIBRARY_PATH = ""
- Report = *tool-install-dir*/`reports`
- Report format = `html`
- PATH
  - SPARC PATH = "`/sbin:/bin:/usr/bin:/usr/bin/sparcv9:/bin/sparcv9`"
  - x64 PATH = "`/sbin:/bin:/usr/bin:/usr/bin/amd64:/bin/amd64`"
- Defined Macros = ""
- System Include Directories = `"/usr/include"`

Installing the Preflight Checker

You can install and run the Preflight Checker on systems starting with Oracle Solaris 10.

How to Install the Preflight Checker

1. **Create a directory for the Preflight Checker.**

   ```
   # mkdir /export/home/preflightcheck/app1
   ```

2. **Download the Oracle Solaris Preflight Checker installable zip file from the Oracle Solaris Preflight Applications Checker download page.**

   - **For SPARC systems, download `PreflightCheckerTool-v11-2-0-SPARC.zip`.**
   - **For x86 systems, download `PreflightCheckerTool-v11-2-0-X86.zip`.**

3. **Unzip the downloaded file in the new directory.**

   For example:

   ```
   # unzip PreflightCheckerTool-v11-2-0-SPARC.zip
   ```

4. **Run the setup script to install the application and provide the location where you want to save the final report.**

   ```
   # cd /export/home/preflightcheck/app1/scripts
   # chmod +x setup.sh
   ```

```
# ./setup.sh
```

If you do not indicate a location where you want to save the final report, during the installation, the final report is saved in the default directory, `/export/home/preflightcheck/app1/reports`.

The script unpacks and installs the tool.

5. **Read the documentation that is downloaded with the application in the `Docs` subdirectory.**

Using the Preflight Checker

You run the Binary Analyzer, Source Code Analyzer, or Runtime Analyzer modules by indicating the directories containing the binaries, source code, or processes running on the Oracle Solaris 10 system that you want to check.

The Preflight Checker can analyze running application processes, application binaries, and application source code on Oracle Solaris 10 and report known potential issues with running the application on Oracle Solaris 11.

Detailed instructions for installing and running the Preflight Checker are available in the `README` file in the `docs` subdirectory of the tools installation directory.

Running the Preflight Checker Application at the Command Line

Once the Preflight Checker is installed, you can start it by using the following command:

```
$ preflightchecker.sh [options] file-or-directory
```

To list the options available for executing the Preflight Checker at the command line (`preflightchecker.sh`), use the `-h` option.

```
$ preflightchecker.sh -h
```

During the static binary analysis, the Preflight Checker locates the object files in the source and checks for the use of deprecated, unsupported, and unstable APIs that might not work on Oracle Solaris 11.

Running Preflight Checker Modules From the GUI

This section provides information about how to use the different modules available in the Preflight Checker from the graphical user interface (GUI).

▼ How to Use the Binary Scanner From the GUI

1. **Launch the Preflight Checker GUI.**

   ```
   # tool-install-dir/bin/preflightcheckerGUI.sh
   ```

   Provide the application information such as application name and version.

   You can also click the More Details button to add more details about the application such as database version. Click the Next button to go to the next window.

2. **Select the Application Checker.**

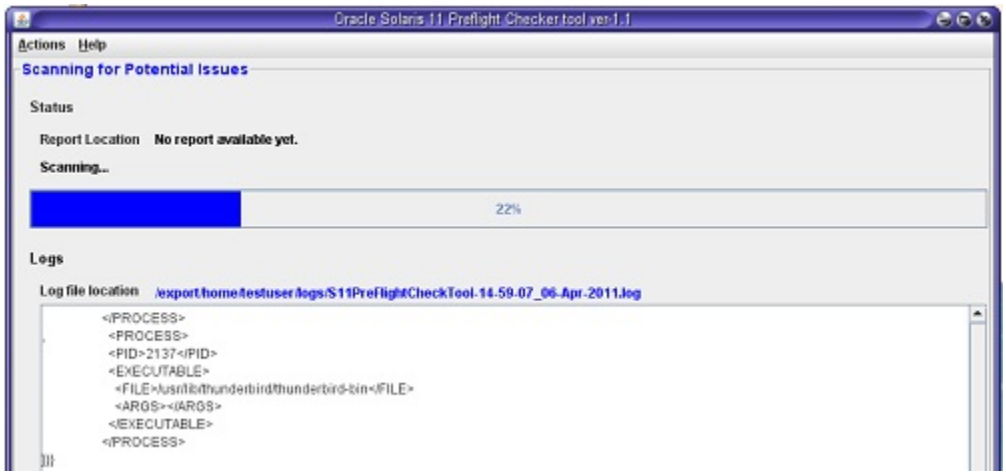
3. **Select the Binary Scanner option.**

4. **In the Summary screen, provide the application details such as the application name, version, path, and `LD_LIBRARY_PATH`.**

5. **Click the Add a Directory button to add the location of the application to be scanned.**

6. **Click the Specify Directory button to choose the folder where you want to save the scan report.**

7. **Click the Start Analysis button to start the binary scan.**



On the completion of binary scanning, the scanner generates a report.

8. **(Optional) To run the test again, click the Re-run S11preflightcheck button.**

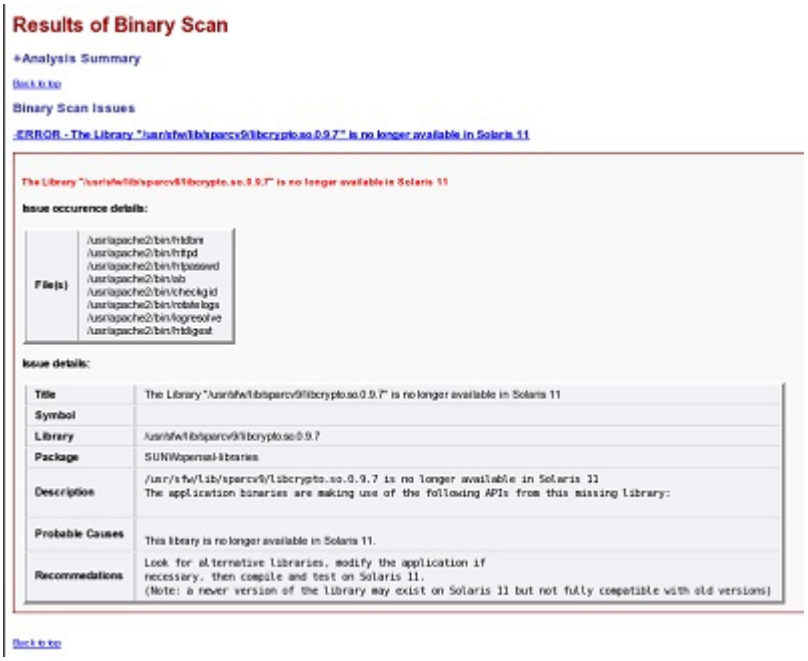
   The current values are used unless you change them.

9. **Click the report location to open the report.**

10. **Click the Analysis Summary link in the Table of contents to display an analysis of the scan.**

11. To display the details of an error or warning, click its link.

The following example shows a sample issue description.



▼ How to Use the Source Code Scanner From the GUI

1. **Launch the Preflight Checker GUI.**

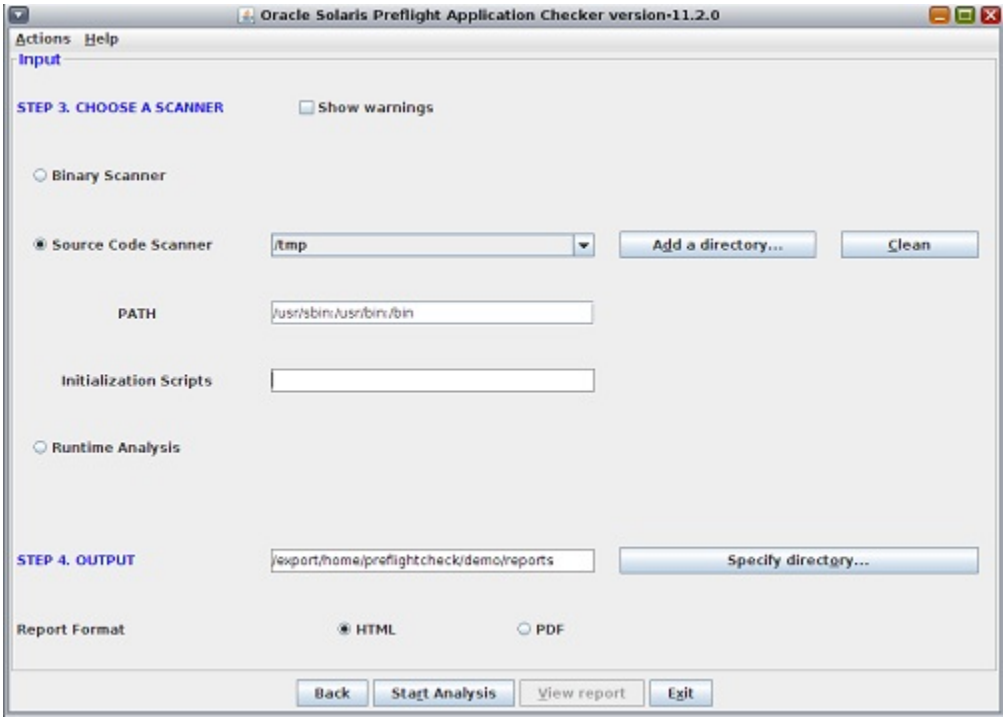
   # *tool-install-dir*`/bin/preflightcheckerGUI.sh`

2. **Select the Application Checker.**

3. **Select the Source Code Scanner option.**

4. **Click the Add a Directory button to add the location of the application to be scanned.**

   The PATH field indicates where to find system executables and user-created executables. As shown in the figure, `/usr/sbin`, `/usr/bin`, and `/bin` are already entered. You can append more locations to the PATH value with a colon (`:`) separator.



5. **Click the Specify Directory button to choose the folder where you want to save the scan report.**

6. **Click the Start Analysis button to start the scan.**

On the completion of source code scanning, the scanner generates a report.

7. **(Optional) To run the test again, click the Re-run S11preflightcheck button.**

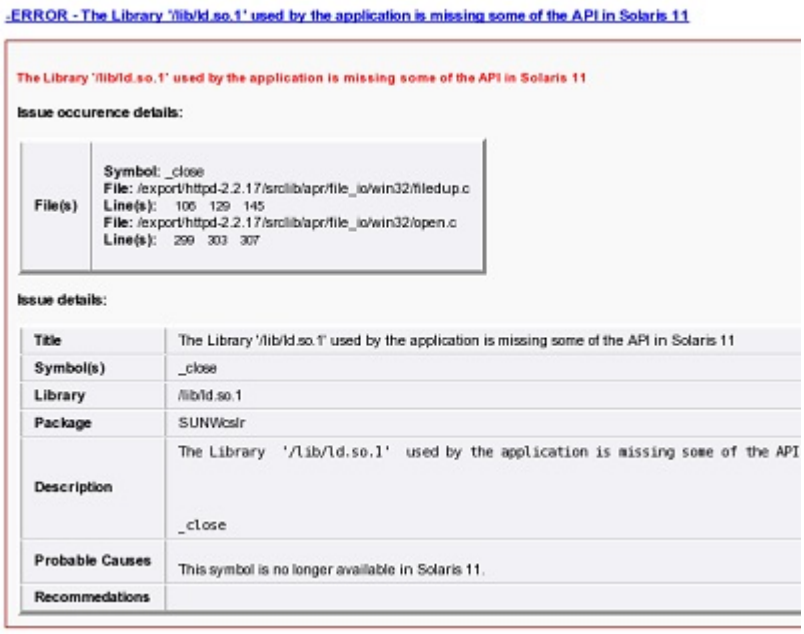
   The current values are used unless you change them.

8. **Click the report location to open the report.**

9. **Click the Analysis Summary link in the Table of contents to display an analysis of the scan.**

10. **To display the details of an error or warning, click its link.**

    The following example shows a sample issue description.



Using the Runtime Analyzer From the GUI

This section provides information about the prerequisites for using the Runtime Analyzer and how to launch it from the GUI.

How to Set Parameters for the Runtime Analyzer

Before You Begin You must have the following Oracle Solaris privileges:

- `basic`
- `dtrace_user`
- `dtrace_proc`
- `dtrace_kernel`

Caution - Do not give the `proc_owner` privilege to the user. Adding this privilege to the user, enables the user to collect the information from processes owned by all other users on the system.

1. **As `root`, run the script to set the parameters.**

   ```
   # /export/home/preflightcheck/app1/scripts/setPriv.sh username
   ```

2. **Set the DTrace value by adding the following line in the `/etc/system` file.**

   ```
   set dtrace:dtrace_dof_maxsize=0x800000
   ```

3. **Set the `fasttrap` value by adding the following line in the `/kernel/drv/fasttrap.conf` file.**

   ```
   fasttrap-max-probes=2500000;
   fasttrap-hash-size=65535;
   ```

4. **Reboot the system after setting the kernel parameters.**

How to Run the Runtime Analyzer From the GUI

1. **Launch the Preflight Checker GUI.**

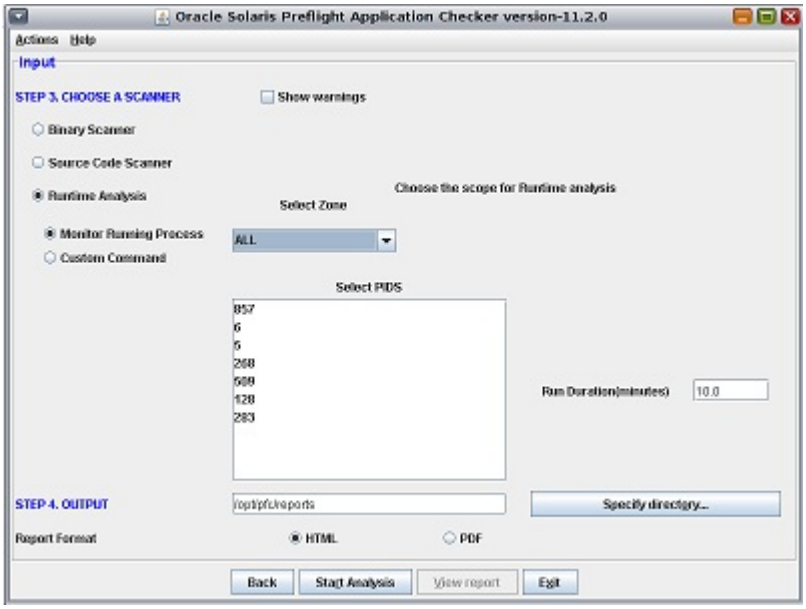
   ```
   # tool-install-dir/bin/preflightcheckerGUI.sh
   ```

2. **Select the Application Checker.**

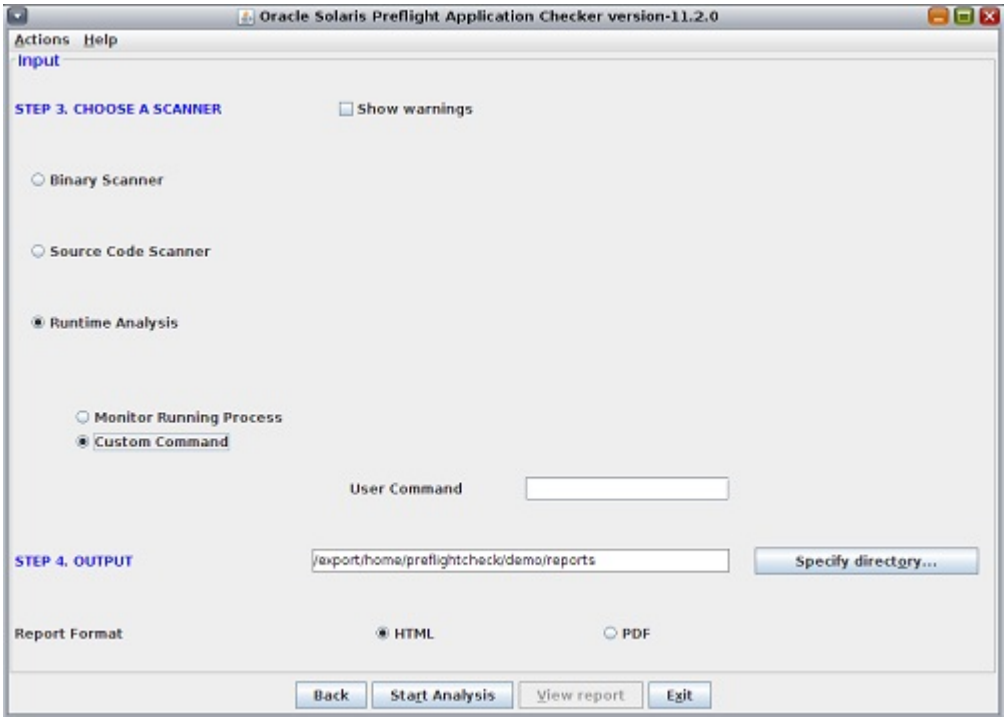
3. **Select the Runtime Analysis option.**

4. **Set the scope for the analysis.**

   - **To monitor a specific running process, select the Monitor Running Process option and select the target `PID` from the list.**



- **To analyze a user command, select the Custom Command option and type the complete command for starting the application to monitor the process.**



5. **(Optional) Set the time for Preflight Checker to scan the application by using the Run Duration field.**

   The Preflight Checker will run for the given time and generate a report based on the scan.

6. **Click the Specify Directory button to choose the folder where you want to save the scan report.**

7. **Click the Start Analysis button to start the scan.**

8. **(Optional) To run the test again, click the Re-run S11preflightcheck button.**

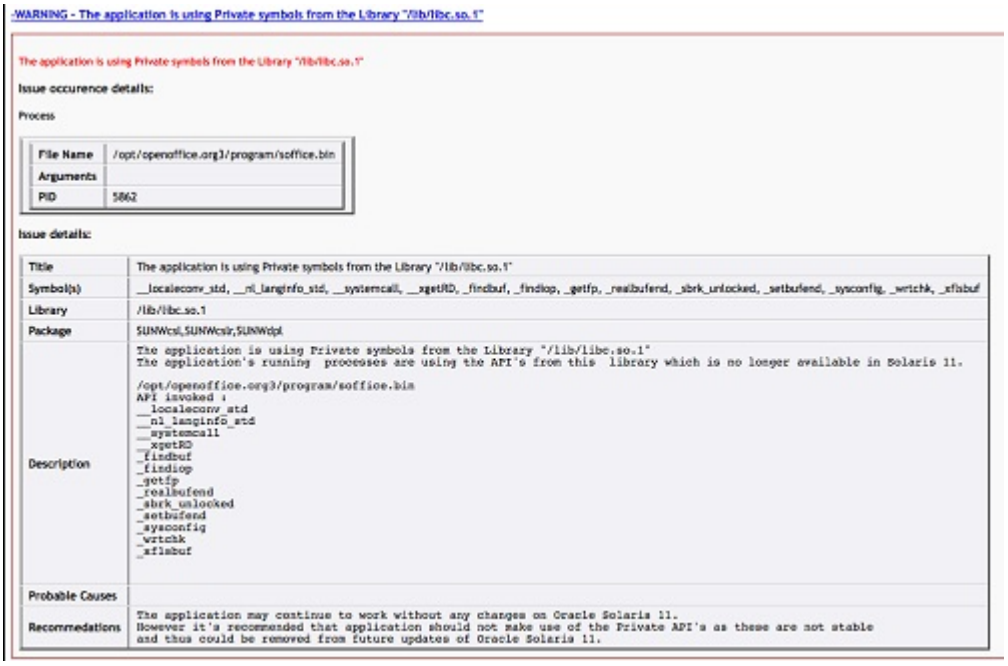
   The current values are used unless you change them.

9. **Click the report location to open the report.**

10. **Click the Analysis Summary link in the Table of contents to display an analysis of the scan.**

11. **To display the details of an error or warning, click its link.**

    The following example shows a sample issue description.



Kernel Checker

The Kernel Checker analyzes kernel modules or device driver binaries and their source code, and reports known potential compliance issues with Oracle Solaris 11.

When running the Binary Analyzer or Source Code Analyzer, you indicate the directories containing the binaries and source code on the Oracle Solaris 10 system that you want to check.

Running the Preflight Kernel Checker at the Command Line

To launch the Kernel Checker, issue the following command:

```
$ ./preflightchecker.sh -kernelchecker [options] file-or-directory
```

To list the options available for executing the Preflight Kernel Checker (`kernelchecker.sh`), use the `-h` option.

```
$ ./preflightchecker.sh -kernelchecker -h
```

During static binary analysis the Kernel Checker recursively searches the directories to locate the object files, and checks for the use of deprecated, unsupported, and unstable APIs which that might not work on Oracle Solaris 11.

▼ How to Run the Kernel Checker From the GUI

1. **Launch the Kernel Checker GUI.**

   ```
   # tool-install-dir/bin/preflightcheckerGUI.sh -kernelchecker
   ```

2. **Select the Kernel Checker option.**

3. **In the Summary screen, provide the application details such as the application name, version, path, and macro definitions.**

4. **Choose whether you want to check the kernel modules or device drivers.**

   - **To check whether the kernel modules will run on Oracle Solaris 11 without any issues, select the Check for Compatibility option.**
   - **To check the device drivers for compliance, select the Check for DDI/DKI Compliance option.**

5. **Click the Specify Directory button to choose the folder where you want to save the scan report.**

6. **Click the Start Analysis button to start the scan.**
   On the completion of kernel scanning, the scanner generates a report.

7. **(Optional) To run the test again, click the Re-run S11preflightcheck button.**
   The current values are used unless you change them.

8. **To open the report, click the View Report button.**

Application Analyzer

The Application Analyzer checks the application for suboptimal coding or implementation practices and the use of specific Oracle Solaris features to get the most out of the Oracle Solaris 11 platform.

Application Analyzer also scans the application to check whether any code change could lead to an immediate performance benefit. This tool analyzes the application's processes and source code and generates a recommendation report. Possible recommendations include migrating from the `gcc` compiler to Oracle Developer Studio, using correct compiler flags to optimize applications, using the Oracle Developer Studio high-performance library, and increasing the performance of servers by using on-chip hardware cryptography.

The Application Analyzer analyzes the application for the following categories of issues:

- Crypto – The Crypto module of the Application Analyzer detects the use of crypto features on the running application processes. For Java applications, the analyzer performs a check for the use of the Java JCE APIs. For native applications, the analyzer checks for use of known list of `crypto` libraries. In both cases, the analyzer provides recommendations to leverage the hardware crypto feature available in SPARC systems.
- Makefile – The Makefile source code scanner scans the application source code and generates a recommendation report in HTML.

  The report contains the following information:

  - If the application is currently using `gcc` compilers for compilation, the report indicates the flag that must be changed for successful compilation using Oracle Developer Studio compilers.
  - If the application is using an older version of Oracle Developer Studio compilers such as forte 6.x or Studio 10, the report indicates the new flag that must be used with latest Oracle Developer Studio compilers for better performance on Oracle Solaris 11. This tool also reports the use of deprecated or unsupported flags from older compilers.
  - Recommendations for compiler optimization flags suitable for better performance of the application on Oracle Solaris 11 based on make files and source code.
- High-Performance Libraries – This module scans the running application processes and generates a recommendation report on the suitability of using specific Oracle Solaris 11 libraries. For example, Sun Performance Library is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other numerically intensive problems. This tool detects the use of such public domain subroutines by running application processes and suggests a corresponding Sun Performance Library.

Running the Application Analyzer at the Command Line

To launch the Application Analyzer, issue the following command:

```
$ appAnalyser.sh [options] file-or-directory
```

To list the options available for executing the Preflight Application Analyzer (`appanalyser.sh`), use the `-h` option:

```
$ appAnalyser.sh -h
```

▼ How to Run the Application Analyzer From the GUI

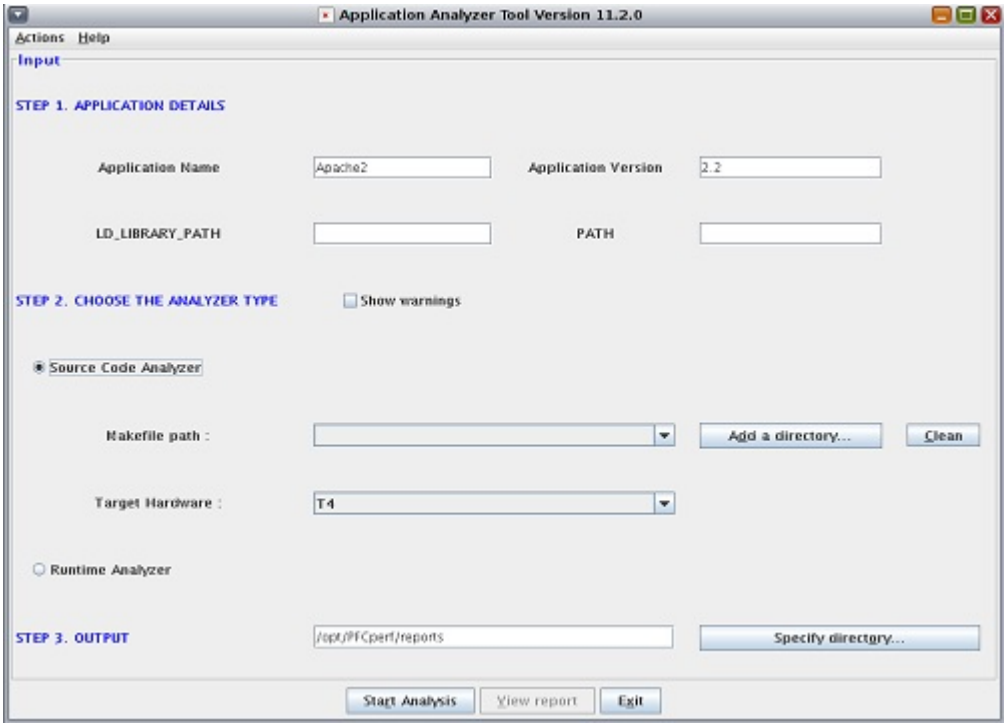
1. **Launch the Application Analyzer GUI.**

   ```
   $ tool-install-dir/bin/appAnalyzerGUI.sh
   ```

2. **Provide the application details such as the application name, version, and path.**

3. **Choose the type of analyzer.**

   - **To check the use of removed, renamed, or modified library functions, select the Source Code Analyzer option.**
   - **To analyze the source code within a scope, such as a zone, or a specific process for a given executable name, select the Runtime Analyzer option.**



4. **Click the Specify Directory button to choose the folder where you want to save the scan report.**

5. **Click the Start Analysis button to start the scan.**

   On the completion of application scanning, the scanner generates a report.

6. **(Optional) To run the test again, click the Re-run S11preflightcheck button.**

   The current values are used unless you change them.

7. **To open the report, click the View Report button.**

Help and Support for Preflight Applications Checker

For more information about updates and the latest version of the Oracle Solaris Preflight Applications Checker, see the Oracle Solaris Preflight Applications Checker 11.3 product page.

Independent Software Vendors (ISV) can get help or support for this tool by contacting `isvsupport_ww@oracle.com`.

Index

P

R

S