Using the Solaris Binary Distributions

This page provides important information for users of the Ice binary distributions for Solaris. You can obtain these distributions at the ZeroC web site.

On this page:

Overview of the Solaris binary distributions

The binary distributions of Ice for Solaris 10 on SPARC and x86/x64 include the following components:

- The Ice run time, including executables for the Ice services and Slice files.
- Run-time libraries for C++ and Java. These libraries enable you to execute Ice applications.
- Tools and libraries for developing Ice applications.

The distributions contain executables and libraries in both 32-bit and 64-bit format. The 32-bit executables are in the `bin` directory and the 32-bit libraries are in the `lib` directory. The 64-bit executables are located in `bin/sparcv9` on SPARC and `bin/amd64` on x64, and the 64-bit libraries are in `lib/sparcv9` on SPARC and `lib/amd64` on x64.

The C++ binaries were created with the following compilers:

- on SPARC, with GCC for Sun Systems 4.3.2
- on x86/x64, with GCC 3.4.3, included in the Solaris 10 x86 distribution

The following Solaris 10 binary packages are required to use these distributions:

- `bzip2` (Solaris package `SUNWbzip`)
- `libexpat` (Solaris package `SUNWlexpt`)
- `openssl` (Solaris packages `SUNWopenssl-libraries` and `SUNWopenssl-commands`)

We also recommend that you install the Solaris 10 Encryption Kit (Solaris packages `SUNWcry` and `SUNWcryr`).

Ice for Java requires J2SE 1.5.0 or later; Java binary packages can be obtained from Oracle.

Setting up your Solaris environment to use Ice

After installing Ice, read the relevant language-specific sections below to learn how to configure your environment and start programming with Ice.

General requirements

In order to use Ice services and tools such as Slice translators, you need to add the location of the Ice binaries to your `PATH` as shown in the bash command below:

```
$ export PATH=<Ice installation directory>/bin:$PATH
```

To use the 64-bit version of these services and tools, add the appropriate architecture-specific subdirectory to `PATH`, making sure it appears before `bi`. For example, on SPARC:

```
$ export PATH=<Ice installation directory>/bin:$PATH
$ export PATH=<Ice installation directory>/bin/sparcv9:$PATH
```

And on x86:

```
$ export PATH=<Ice installation directory>/bin:$PATH
$ export PATH=<Ice installation directory>/bin/amd64:$PATH
```

Ice shared libraries and executables in this distribution contain `/opt/Ice-3.4/lib` as the embedded runpath (the runpath for 64-bit libraries and executables is `/opt/Ice-3.4/lib/sparcv9` or `/opt/Ice-3.4/lib/amd64`). In order to run Ice services and tools, you can do one of the following:

- Create a symbolic link `/opt/Ice-3.4` that points to your Ice installation:

  ```
  $ ln -s <Ice installation directory> /opt/Ice-3.4
  ```

- Add the Ice `lib` directory to your `LD_LIBRARY_PATH` and `LD_LIBRARY_PATH_64` environment variables. For example, on SPARC:

  ```
  $ export LD_LIBRARY_PATH=<Ice installation directory>/lib:$LD_LIBRARY_PATH
  $ export LD_LIBRARY_PATH_64=<Ice installation directory>/lib/sparcv9:$LD_LIBRARY_PATH_64
  ```

  And on x86:

  ```
  $ export LD_LIBRARY_PATH=<Ice installation directory>/lib:$LD_LIBRARY_PATH
  $ export LD_LIBRARY_PATH_64=<Ice installation directory>/lib/amd64:$LD_LIBRARY_PATH_64
  ```

C++

When compiling Ice for C++ programs, you must pass the Ice include directory to the compiler with the `-I` option, and the Ice library directory with the `-L` option. Furthermore, a C++ program needs to link with at least `libIce` and `libIceUtil`, so a typical link command would look like this:

```
$ CC -I <Ice installation directory>/include -o myprogram myprogram.o \
  -L<Ice installation directory>/lib -lIce -lIceUtil
```

Additional libraries are necessary if you are using an Ice service such as IceGrid or Glacier2.

Java

To use Ice for Java, you must add `Ice.jar` to your `CLASSPATH`, as shown below:

```
$ export CLASSPATH=<Ice installation directory>/lib/Ice.jar:$CLASSPATH
```

If you intend to use Freeze for Java, you must include `Freeze.jar` in your `CLASSPATH` along with `Ice.jar`:

```
$ export CLASSPATH=<Ice installation directory>/lib/Freeze.jar:$CLASSPATH
```

Note that Freeze requires Berkeley DB. Freeze.jar contains a manifest that automatically loads the Berkeley DB classes (`db.jar`), which means you do not need to include this file in your `CLASSPATH` when executing a Freeze application. However the JVM does require that the directory containing Berkeley DB's native libraries be listed in `java.library.path`. Modifying your `LD_LIBRARY_PATH` (or `LD_LIBRARY_PATH_64`, if you use a 64-bit JVM) as described above satisfies this requirement.

When building a Java application that uses Freeze, you will need to add the Berkeley DB JAR file to your `CLASSPATH`:

```
$ export CLASSPATH=<Ice installation directory>/lib/db.jar:$CLASSPATH
```

Ice includes ant tasks for translating Slice to Java. The ant tasks allow `slice2java` and `slice2freezej` to be invoked from the ant build system. These tasks require one of the following:

- Specify the location of the Ice installation containing the translators with the `ice.home` property:

```
ant -Dice.home=/home/bill/Ice-3.4.2
```

- Set the `ICE_HOME` environment variable to specify the location of the Ice installation containing the translators:

```
$ export ICE_HOME=/home/bill/Ice-3.4.2
```

- If neither `ice.home` nor `ICE_HOME` is available, the ant tasks will simply invoke the translator without an absolute path, relying on the translators being in a directory in your `PATH` for successful execution.

Ice for Java supports protocol compression using the bzip2 classes included with ant. Compression is automatically enabled if these classes are present in your `CLASSPATH`. You can either add `ant.jar` to your `CLASSPATH`, or download only the bzip2 classes from:

http://www.kohsuke.org/bzip2/

Note that these classes are a pure Java implementation of the bzip2 algorithm and therefore add significant latency to Ice requests.

Starting the IceGrid Administrative Console

A Java-based graphical tool for administering IceGrid applications is included in these distributions. The Java archive file is installed as

<Ice installation directory>`/lib/IceGridGUI.jar`

With a suitable Java environment, you can execute the application using the following command:

```
$ java -jar IceGridGUI.jar
```

Using the sample programs on Solaris

Sample programs are provided in a separate archive, which can be downloaded from the ZeroC web site.

Please refer to the `README.DEMOS` file included in that archive for more information.

Python support on Solaris

Ice for Python is fully supported on Solaris. However it is not included in this distribution because there is no suitable binary distribution for Python itself available from Oracle or Sunfreeware.

If you want to use Ice for Python on Solaris, you can build a source distribution available at the ZeroC web site.

Third-party packages for Solaris

The binary distributions for Solaris include the following third-party packages as separate binary libraries:

- Berkeley DB 4.8.30 (C/C++ and Java run time)
- QtCore and QtSql 4.5.3 with SQLite and PostreSQL drivers built-in (C++ run time)

The `QtCore` and `QtSql` libraries in the Solaris/SPARC distribution are only provided in 32 bits due to issues compiling Qt in 64 bits with GCC 4.3.2.