Using the OS X Binary Distribution

This page provides important information for users of the Ice binary distribution for OS X. You can obtain this distribution at the ZeroC web site.

On this page:

- Overview of the OS X binary distribution
- Setting up your OS X environment to use Ice
  - General requirements
  - C++
  - Java
  - Android
  - Python
- Using the sample programs on OS X
- Third-party packages for OS X

Overview of the OS X binary distribution

The binary distribution of Ice for OS X includes the following components:

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

The binary distribution was compiled on OS X 10.8 using Xcode 4.6. The binaries in this distribution are fat binaries with support for both Intel 32-bit and Intel 64-bit architectures.

The Ice extension for Python included in this distribution requires Python 2.7.2, installed with OS X 10.8.

The binaries are installed at the following locations:

- Ice in `/Library/Developer/Ice-3.5.1`
- IceGrid Administrative Console in `/Applications`
- Ice for Python in `/Library/Python/2.7/site-packages`

Setting up your OS X 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
```

If you run applications that load the IceSSL plug-in, you have to do one of the following in order to correctly load IceSSL:

- Set the `DYLD_LIBRARY_PATH` environment variable:

  ```
  $ export DYLD_LIBRARY_PATH=<Ice installation directory>/lib:$DYLD_LIBRARY_PATH
  ```

- Use a relative or absolute path for the IceSSL library in the IceSSL configuration:

  ```
  Ice.Plugin.IceSSL=<Ice installation directory>/lib/IceSSL:createIceSSL
  ```

For C++11 builds you must use `<Ice installation directory>/lib/c++11` instead of `<Ice installation directory>/lib`.

C++

The Ice binary distribution for OS X includes two sets of libraries built with and without C++11 support. C++11 enabled libraries are installed in `<Ice installation directory>/lib/c++11` and use LLVM `libc++`, while the libraries in `<Ice installation directory>/lib` use `libstdc++` and do not support C++11 features.

When compiling Ice for C++ programs, you must pass the `-pthread` option and a `-I` option specifying the Ice include directory. A typical compile command would look like this:

```
$ c++ -I <Ice installation directory>/include -c -pthread myprogram.cpp
```

When linking a program you must pass the Ice library directory with the `-L` option and set the program run path using the `-rpath` linker option. Furthermore, a C++ program needs to link with at least `libIce` and `libIceUtil`. A typical link command would look like this:

```
$ c++ -o myprogram myprogram.o -Wl,-rpath,<Ice installation directory>/lib -L<Ice installation directory>
/lib -lIce -lIceUtil
```

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

To build fat binaries or binaries using an architecture that differs from the default architecture, you can specify the Clang `-arch` compiler flag. For example, use `-arch i386 -arch x86_64` to build Intel 32-bit and 64-bit fat binaries.

As shown in the previous example, we need to set the application run path when we link the application. This is required because Ice libraries use `@rpath` as the prefix in their install names, which allows the application developer to relocate the Ice libraries without needing to rebuild them. You should use the `/Library/Developer/Ice-3.5` symbolic link for the run path if you want your executables to automatically use new Ice patch releases. This symbolic link is updated by the Ice installer to point to the latest installed Ice version.

If you want to include Ice in your application bundle, you will need to copy the necessary Ice libraries to the `Contents/Frameworks` subdirectory of your bundle and use `@loader_path/../Frameworks` as the run path when linking the application.

Please refer to the `dyld` man page on your OS X system to learn more about `@rpath` and `@loader_path`.

For C++11 builds you must use `<Ice installation directory>/lib/c++11` instead of `<Ice installation directory>/lib`.

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` and `db.jar` in your `CLASSPATH` along with `Ice.jar`:

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

Classes for the other Ice services are provided in separate JAR files:

- `Glacier2.jar`
- `IceBox.jar`
- `IceGrid.jar`
- `IcePatch2.jar`
- `IceStorm.jar`

If your application uses any of these services, add the appropriate JAR files to your `CLASSPATH` as shown above.

The JVM also requires that the directory containing Berkeley DB's native libraries be listed in `java.library.path`, therefore you must add this directory to your `DYLD_LIBRARY_PATH`.

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=<Ice installation directory>
```

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

```
$ export ICE_HOME=<Ice installation directory>
```

- 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.

Eclipse Development

ZeroC has created a Slice2Java plug-in for Eclipse that automates the translation of your Slice files. If you use Eclipse, we strongly recommend using this plug-in for your own development.

The Slice2Java plug-in is required if you intend to build any of the Android projects included in the sample programs.

For installation instructions, please refer to the ZeroC web site. The manual provides more information about configuring the plug-in and using it in your projects.

Android

Ice requires Android 2.3 or later. Aside from that, there are no other special requirements for using Ice in an Android application. We strongly recommend installing our Slice2Java plug-in for Eclipse to automate the compilation of your Slice definitions.

Python

The Ice for Python run-time installs the Ice extension and its associated Python files into the `site-packages` directory (`/Library/Python/2.7/site-packages`).

Using the sample programs on OS X

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

Third-party packages for OS X

The binary distribution for OS X includes the following third-party packages as separate binary libraries:

- Berkeley DB 5.3.21 (C/C++ and Java run time)