Building Ice for Java

This page describes how to build and install Ice for Java from source code. If you prefer, you can also download binary distributions for the supported platforms.

On this page:

- Java Build Requirements
  - Operating Systems
  - Java Version
  - Slice to Java Translator
  - Berkeley DB
  - Bzip2 Compression
  - JGoodies
  - ProGuard
  - Java Application Bundler
  - Android
- Compiling Ice for Java
  - Preparing to Build
  - Building Ice for Java
  - Ant Tasks
- Installing Ice for Java
- Eclipse Development
- Using Ice in Android
  - Android Sample Projects
  - Android Tests
- Running Java Tests and Demos
- IceGrid Administrative Console

Java Build Requirements

Operating Systems

Ice for Java is expected to build and run properly on Windows, OS X, Solaris, and any recent Linux distribution for x86 and x86_64, and was extensively tested using the operating systems and Java versions listed on our platforms page. Due to the portability of Java, it is very likely that it will also work on other platforms for which a suitable Java implementation is available.

Note, however, that you will need the Slice to Java translator. ZeroC provides translator binaries for our supported platforms. For other platforms, you will have to either port Ice for C++ (which contains the Slice to Java translator), or you will have to translate your Slice files to Java on a supported platform, and copy the generated Java files to your target platform.

Java Version

Ice for Java requires J2SE 1.6.0 or later.

The Metrics Graph feature of the graphical IceGrid administrative tool requires J2SE 7u6 or later with JavaFX support. This feature will not be available if you build the source with a JVM that lacks support for JavaFX. Alternatively, building the source using J2SE 7u6 or later with JavaFX produces a JAR file that can be used in JVMs with or without JavaFX support, as the Metrics Graph feature is enabled dynamically.

Make sure that the `javac` and `java` commands are present in your `PATH`.

Slice to Java Translator

You will need the Slice to Java translator and supporting executables and libraries. You can download a binary distribution from the ZeroC web site, or you can build Ice for C++ yourself.

Berkeley DB

"Freeze" is an optional Ice component that provides a persistence facility for Ice applications. Freeze uses Berkeley DB as its underlying database and currently requires Berkeley DB version 5.3 (the recommended version is 5.3.21).

ZeroC includes Berkeley DB in the binary distributions for all supported platforms. If you would rather build Berkeley DB yourself, a source distribution with build instructions is also available.

In order to run an application that uses Freeze, you must add `db.jar` to your `CLASSPATH` and verify that the Berkeley DB shared libraries are in your `java.library.path`. On Linux, this can be achieved by adding *`Berkeley DB home`*`/lib` to your `LD_LIBRARY_PATH`, for example:

```
$ export LD_LIBRARY_PATH=/opt/db53/lib:$LD_LIBRARY_PATH
```

Bzip2 Compression

Ice for Java supports protocol compression using the bzip2 classes included with Apache Ant or available separately from kohsuke.org.

Compression is automatically enabled if the classes are present in the `CLASSPATH`. To use the classes included with Ant, simply add `ant.jar` to your `CLASSPATH`.

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

JGoodies

The graphical IceGrid administrative tool uses the JGoodies libraries Common, Forms, and Looks. The following versions were tested:

- JGoodies Common 1.4.0
- JGoodies Forms 1.6.0
- JGoodies Looks 2.5.2

ProGuard

ProGuard is used to create the standalone JAR file for the graphical IceGrid administrative tool.

Ice for Java has been tested with ProGuard 4.8.

Java Application Bundler

The Java Application Bundler is used to create the OS X application bundle for the graphical IceGrid administrative tool.

Android

The Ice run time (`lib/Ice.jar`) is fully compatible with Android 2.3 or later. Refer to the sections that discuss Eclipse and Android for additional details.

Compiling Ice for Java

Preparing to Build

The Ice for Java build system requires Apache Ant 1.7.0 or later.

This source distribution cannot be compiled successfully without the Berkeley DB run time for Java (`db.jar`). On Unix platforms, the build system searches for this file in two locations:

- `/usr/share/java/db-5.3.21.jar`
- `/opt/Ice-3.5.1/lib/db.jar`

If neither of these files is present on your system, or if you are not using a Unix platform, you must add `db.jar` to your `CLASSPATH`.

The build system also requires the Slice translators from Ice for C++. If you have not built Ice for C++ in this source distribution, you must set the `ICE_HOME` environment variable with the path name of your Ice installation. For example:

```
# On Unix
$ export ICE_HOME=/opt/Ice-3.5.1 (For local build)
$ export ICE_HOME=/usr (For RPM installation)

# On Windows
> set ICE_HOME=C:\Ice-3.5.1
```

Before building Ice for Java, review the settings in the file `config/build.properties` and edit as necessary.

If you intend to build the IceGrid Administrative Console, you have two options:

1. Build it as a standalone JAR file. In this case, confirm the locations of the JGoodies libraries in `config/build.properties` and add ProGuard to your `CLASSPATH`.
2. Build it as a regular JAR file. The JGoodies libraries must either be present in your `CLASSPATH` or available at the locations defined in `config/build.properties`.

In OS X you also have the option to create an Application Bundle. The Java Application Bundler must be present in your `CLASSPATH`. After the build completes, the application bundle can be found in `lib/IceGrid Admin.app`. During installation the bundle is copied to the `Ice-3.5.1/bin` subdirectory.

Building Ice for Java

To build only the Ice run time (`Ice.jar`), use the following command:

```
ant ice-jar
```

To build the Ice run time along with Freeze, IceBox, IceGrid, IceStorm, IcePatch2 services and the IceGrid administrative console, use this command:

```
ant jar
```

Finally, to build the entire source distribution, including tests and sample programs, use this command:

```
ant
```

Upon completion, the JAR files can be found in the `lib` subdirectory.

If at any time you wish to discard the current build and start a new one, use these commands:

```
ant clean
ant
```

Ant Tasks

The build system uses some Ice-specific Ant tasks for executing the Slice translators that you may find useful in your own Ant projects. The Java classes for the tasks are supplied in `lib/ant-ice.jar`. You can use an Ant project file from any of the sample programs as a guide for using the tasks, or you can review the source code in the `src/ant` subdirectory.

Installing Ice for Java

Run `ant install` to install Ice for Java in the directory specified by the `prefix` variable in `config/build.properties`.

> It is not necessary to rebuild the source after changing the value of the `prefix` variable.

After installation, add `Ice.jar` to your `CLASSPATH`. If you plan to use other Ice services, you must also add the appropriate JAR files to your `CLASSPATH`:

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

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 installing this plug-in to simplify your Ice projects.

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

Using Ice in 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.

Android Sample Projects

Several sample Android projects are provided in the `demo/android` subdirectory. You must use Eclipse and the Slice2Java plug-in to build these projects.

In Eclipse, you can open a sample project by choosing **File->Import...**; in the "General" group, select "Existing Project into Workspace", then open one of the subdirectories of `demo/android`.

The sample projects are configured to locate the Ice run time JAR file (`Ice.jar`) via the `ICE_JAR_HOME` classpath variable.

If you installed `Ice.jar` in a different location, you will need to add it as an external JAR file in each sample project:

1. Open the project's properties and select Java Build Path
2. Click on the Libraries tab
3. Click Add External JARs... and navigate to `Ice.jar`
4. Click OK to save your settings

Android Tests

The Ice test suite for Android consists of a JAR file containing the individual test cases and an Android app that loads and runs these tests.

You must first build the test JAR file:

```
ant test-android-jar
```

This produces `IceAndroidTest.jar`. Next, you must use Eclipse and the Slice2Java plug-in to build the the test app. In Eclipse, open the test project by choosing **File->Import...**; in the "General" group, select "Existing Project into Workspace", then navigate to the `test/android` directory.

The test project is configured to locate the Ice run time JAR file (`Ice.jar`) and `IceAndroidTest.jar` via the `ICE_JAR_HOME` classpath variable. The Slice2Java plug-in configures this variable automatically based on your setting for the SDK location in the Eclipse preferences.

After successfully building the project, deploy it onto a suitable emulator or device.

Running Java Tests and Demos

Some of the Ice for Java tests employ applications that are part of the Ice for C++ distribution. If you have not built Ice for C++ in this source distribution then you must set the `ICE_HOME` environment variable with the path name of your Ice installation:

```
# On Unix
$ export ICE_HOME=/opt/Ice-3.5.0 (For local build)
$ export ICE_HOME=/usr (For RPM installation)

# On Windows
> set ICE_HOME=c:\Ice-3.5.0
```

Python is required to run the test suite. To run the tests, open a command window and change to the top-level directory. At the command prompt, execute:

```
python allTests.py
```

You can also run tests individually by changing to the test directory and running this command:

```
python run.py
```

If everything worked out, you should see lots of "ok" messages. In case of a failure, the tests abort with "failed".

The `Glacier2/router` test will fail if it is run with the `--compress` option but the bzip2 classes are not present in your `CLASSPATH`.

If you want to try out any of the demos, make sure to add `lib/Ice.jar` and classes to your `CLASSPATH`.

For demos that use Ice services, you must also add the appropriate service JAR files to your `CLASSPATH`:

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

Change to the desired demo directory and follow the instructions in the `README` file. If no `README` file is present, the demo can be run by entering the following command to start the server:

```
java Server
```

Then in a separate window enter this command to start the client:

```
java Client
```

IceGrid Administrative Console

Ice for Java includes a graphical administrative console for IceGrid. If you elected to build the console, it can be found in the file `lib/IceGridGUI.jar`.

If you built this JAR file using ProGuard, it is completely self-contained and has no external dependencies, in which case you can start the console with the following command:

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

If you compiled the console without ProGuard, you will need to add IceGridGUI.jar to your CLASSPATH. You can start the console with this command:

```
java IceGridGUI.Main
```

In OS X there is also an application bundle named *IceGrid Admin*. You can start the IceGrid Administrative Console by double-clicking the *IceGrid Admin* icon in Finder.