Using the Linux Binary Distributions

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

On this page:

Overview of the Ice Binary Distributions for Linux

RPM Packages

ZeroC provides the following RPMs for Red Hat Enterprise Linux, SuSE Linux Enterprise Server, and Amazon Linux:

RPM	Description
`ice`	Slice files and related documentation
`ice-c++-devel`	C++ header files, libraries, and Slice compilers
`ice-java`	Java run time JAR files
`ice-java-devel`	Slice compilers and Ant tasks for Java development
`ice-libs`	C++ run-time libraries
`ice-mono`	Mono run-time assemblies
`ice-mono-devel`	Slice compiler and libraries for Mono development
`ice-php`	PHP extension and run time files
`ice-php-devel`	Slice compiler for PHP development
`ice-python`	Python extension and run time files
`ice-python-devel`	Slice compiler for Python development
`ice-ruby`	Ruby extension and run time files
`ice-ruby-devel`	Slice compiler for Ruby development
`ice-servers`	Server executables and sample `init.d` scripts
`ice-utils`	Utilities necessary for administering an Ice installation

The Mono RPMs are currently available only for SuSE Enterprise Linux Server.

ZeroC also supplies RPMs for the following third-party packages:

RPM	Description
`db53`	Berkeley DB 5.3.21 C and C++ run time libraries

`db53-devel`	C++ development files for Berkeley DB 5.3.21
`db53-java`	Berkeley DB 5.3.21 Java run time
`db53-utils`	Berkeley DB 5.3.21 command-line utilities
`mcpp-devel`	MCPP C++ preprocessor library

The `db53-devel` and the `mcpp-devel` RPMs are only necessary for building Ice from source.

DEB Packages

ZeroC provides the following DEB packages for Ubuntu 14.04:

Package	Description
`ice`	Meta package that installs all run-time packages, servers and utilities
`ice-dev`	Meta package that installs all development packages
`ice3.5-common`	Slice files and related documentation
`libice3.5++`	C++ run-time libraries
`libice++-dev`	C++ header files, libraries, and Slice compiler
`libice3.5-java`	Java run-time libraries
`libice-java-dev`	Slice compilers and Ant tasks for Java development
`libice3.5-cil`	Mono run-time assemblies
`libice-cil-dev`	Slice compiler and libraries for Mono development
`libice3.5-php`	PHP extension and run time files
`libice-php-dev`	Slice compiler for PHP development
`libice3.5-python`	Python extension and run time files
`libice-python-dev`	Slice compiler for Python development
`libice3.5-ruby`	Ruby extension and run time files
`libice-ruby-dev`	Slice compiler for Ruby development
`ice-utils`	Utilities necessary for administering an Ice installation
`icebox`	IceBox server for C++
`icebox-cil`	IceBox server for Mono
`icegrid`	IceGrid service
`libicestorm3.5`	IceStorm service
`glacier2`	Glacier2 service
`icepatch2`	IcePatch2 service
`libfreeze3.5++`	Freeze for C++ run-time libraries
`libfreeze3.5-java`	Freeze for Java run-time library
`libfreeze++-dev`	Freeze for C++ Slice compiler
`libfreeze-java-dev`	Freeze for Java Slice compiler

ZeroC also supplies the following third-party packages:

Package	Description

`libdb5.3`	Berkeley DB 5.3.21 C run-time libraries
`libdb5.3-dev`	C development files for Berkeley DB 5.3.21
`libdb5.3++`	Berkeley DB 5.3.21 C++ run-time libraries
`libdb5.3++-dev`	C++ development files for Berkeley DB 5.3.21
`libdb5.3-java`	Berkeley DB 5.3.21 Java run-time libraries
`db5.3-utils`	Berkeley DB 5.3.21 command-line utilities

The `libdb5.3-dev` and `libdb5.3++-dev` packages are only necessary for building Ice from source.

Setting up your Linux 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.

C++

A C++ program needs to link with at least `libIce` and `libIceUtil`, so a typical link command would look like this:

```
$ CC -o myprogram myprogram.o -pthread -lIce -lIceUtil
```

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

Always use the `-pthread` option when compiling and linking your Ice applications.

The Linux binary distribution is not binary compatible with C++11 objects: you should not build your source files with `-std=c++0x`, `-std=gnu++0x`, `-std=c++11`, `-std=gnu++11`, `-std=c++14`, or `-std=gnu++14`.

In order to use any of these options when compiling your source files, you need to first rebuild Ice C++ from sources in C++11 mode.

Java

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

```
$ export CLASSPATH=/usr/share/java/Ice.jar:$CLASSPATH
```

If you intend to use Freeze for Java, you must include `Freeze.jar` and `db-5.3.21.jar` in your `CLASSPATH` along with `Ice.jar`:

```
$ export CLASSPATH=/usr/share/java/Freeze.jar:$CLASSPATH
$ export CLASSPATH=/usr/share/java/db-5.3.21.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 `LD_LIBRARY_PATH`. Assuming you are using ZeroC's distribution of Berkeley DB, the bash command is shown below:

```
$ export LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH                (RHEL, SLES, Amazon)
$ export LD_LIBRARY_PATH=/usr/lib/i386-linux-gnu:$LD_LIBRARY_PATH  (Ubuntu)
```

On an x86_64 system with a 64-bit JVM, the 64-bit Berkeley DB libraries are installed in a different directory:

```
$ export LD_LIBRARY_PATH=/usr/lib64:$LD_LIBRARY_PATH                (RHEL, SLES, Amazon)
$ export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH  (Ubuntu)
```

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.

When using the Ice for Java SSL plugin (IceSSL), you may experience occasional hangs. The most likely reason is that your system's entropy pool is empty. If you have sufficient system privileges, you can solve this issue by editing the file

<java.home>`/jre/lib/security/java.security`

and changing it to use `/dev/urandom` instead of `/dev/random`. If you do not have permission to modify the security file, you can also use the command-line option shown below:

```
$ java -Djava.security.egd=file:/dev/urandom MyClass ...
```

On Linux systems with IPv6 enabled, you may experience occasional hangs the first time an Ice object adapter is activated within a JVM. A work-around is to disable IPv6 support by setting the Java property `java.net.preferIPv4Stack` to true. For example:

```
$ java -Djava.net.preferIPv4Stack=true MyClass ...
```

For more information on this issue, refer to the relevant Java bug.

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 package installs the Ice extension and its associated Python files into the system packages directory. The installation also includes a `.pth` file that enables you to import the various Ice modules without requiring any additional configuration.

Mono

ZeroC's run-time package adds the .NET run-time libraries to the global assembly cache (GAC), so that no changes to your environment are necessary to locate the assemblies.

The instructions for running the demos assume that you have configured your kernel to automatically execute the Mono interpreter. Visit the Mono Project for a description of how to configure your kernel to register Mono's .exe files as non-native binaries. If you don't want to configure your kernel, you will need to run executables with `mono`. For example,

```
$ mono server.exe
```

Managed code

The main Ice for .NET assembly (`Ice.dll`) included in the distribution uses unmanaged code. If you require only managed code then you can download the Ice source distribution and build Ice for Mono in a purely managed version. Note that the managed version of Ice for Mono omits support for protocol compression.

You can download the source distribution at the ZeroC web site.

Ruby

The Ice for Ruby run-time package installs the Ice extension and its associated Ruby files into the `site_ruby` directory. No additional configuration is necessary to use Ice in your Ruby programs.

PHP

The Ice extension for PHP is loaded automatically when the interpreter loads the contents of the file `/etc/php.d/ice.ini` (on Red Hat Enterprise Linux and Amazon Linux) or `/etc/php5/conf.d/ice.ini` (on SuSE Linux Enterprise Server and Ubuntu). This file contains the line shown below:

```
extension=IcePHP.so
```

You can modify this file to include additional configuration directives.

At run time, the PHP interpreter requires the Ice shared libraries.

You can verify that the Ice extension is installed properly by examining the output of the `php -m` command, or by calling the `phpinfo()` function from a script.

Your application will also need to include at least some of the Ice for PHP run-time source files (installed in `/usr/share/php` on RHEL, Amazon Linux, and Ubuntu, and in `/usr/share/php5` on SLES). This installation directory is included in PHP's default include path, which you can verify by executing the following command:

```
% php -i | grep include_path
```

If the installation directory is listed, no further action is necessary to make the run-time source files available to your application. Otherwise, you can modify the `include_path` setting in `php.ini` to add the installation directory:

```
include_path = /usr/share/php:...
```

Another option is to modify the include path from within your script prior to including any Ice run-time file:

PHP

```
ini_set('include_path', ini_get('include_path') . PATH_SEPARATOR . '/usr/share/php');
require 'Ice.php'; // Load the core Ice run time definitions.
```

SELinux Notes (for Red Hat Enterprise Linux users)

SELinux augments the traditional Unix permissions with a number of new features. In particular, SELinux can prevent the `httpd` daemon from opening network connections and reading files without the proper SELinux types.

If you suspect that your PHP application does not work due to SELinux restrictions, we recommend that you first try it with SELinux disabled. As root, run:

```
# setenforce 0
```

to disable SELinux until the next reboot of your computer.

If you want to run `httpd` with the Ice extension and SELinux enabled, you must do the following:

1. Allow httpd to open network connections:

   ```
   # setsebool httpd_can_network_connect=1
   ```

   You can add the -P option to make this setting persistent across reboots.

1. Make sure any `.ice` file used by your PHP scripts can be read by `httpd`. The enclosing directory also needs to be accessible. For example:

   ```
   # chcon -R -t httpd_sys_content_t /opt/MyApp/slice
   ```

For more information on SELinux in Red Hat Enterprise Linux, refer to this Red Hat document.

Using the sample programs on Linux

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

Startup scripts for IceGrid and Glacier2 services

The distributions include the following sample `/etc/init.d` scripts and associated configuration files:

- `/etc/init.d/icegridregistry` and `/etc/icegridregistry.conf`
- `/etc/init.d/icegridnode` and `/etc/icegridnode.conf`
- `/etc/init.d/glacier2router` and `/etc/glacier2router.conf`

The installation also creates a user account and group for running these services (account `ice` and group `ice`), and data directories for `icegridregistry` and `icegridnode` (`/var/lib/ice/icegrid/registry` and `/var/lib/ice/icegrid/node1`).

By default, all these services are off at all runlevels. You need to manually switch on one or more runlevels, for example:

```
#
# On a Red Hat Enterprise Linux system, configure the icegridregistry
# to start at runlevel 3 and 5:
#
chkconfig --level 35 icegridregistry on

#
# On a SuSE Linux Enterprise Server system, configure the
# icegridregistry to start at runlevel 3 and 5:
#
chkconfig icegridregistry 35

#
# On an Ubuntu system, configure the icegridregistry to start at the
# default run levels:
#
sudo update-rc.d icegridregistry defaults
```

Before doing so, please review the script itself and its associated configuration file.