Building Ice for .NET with Mono

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

On this page:

Mono Build Requirements

Ice for .NET requires Mono version 2.0.1 or later and is expected to build and run properly on any recent Linux distribution for x86 and x86_64 that supports Mono. Ice for .NET has been extensively tested using the operating systems and Mono versions listed on our platforms page.

You need to install the `mono-core` and the `mono-devel` RPMs to build Ice for .NET on SuSE Linux Enterprise Server.

IceSSL is not currently supported for Mono.

Compiling Ice for .NET with Mono

Unpack the source archive. The .NET sources will be located in the `Ice-3.5.1/cs` subdirectory.

You will need the Slice to C# translator (`slice2cs`) and supporting executables and libraries. You can download a binary distribution from the ZeroC web site, or you can build Ice for C++ yourself.

If you have not built Ice for C++ in the `cpp` subdirectory, set `ICE_HOME` to the directory of your Ice for C++ installation. For example:

```
$ export ICE_HOME=/opt/Ice-3.5.1
```

Change to the `cs` subdirectory of the Ice source distribution:

```
$ cd Ice-3.5.1/cs
```

Open `config/Make.rules.cs` and review the comments that describe the settings you can modify. For example, you may wish to enable optimization.

Run make:

```
$ make
```

The tests and sample programs are built automatically. If you modify the source code of a sample program, you can rebuild it using make.

Running the .NET Tests

Some of the Ice for .NET tests employ applications that are part of Ice for C++. If you have not built Ice for C++ from the `cpp` subdirectory, then you need to set the `ICE_HOME` environment variable to the path where these applications are installed for the tests to run properly:

```
$ export ICE_HOME=/opt/Ice-3.5.1
```

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".

Running the .NET Demos

To run the demos, you need to have the `cpp/bin` directory in your `PATH` and the `cs/Assemblies` directory in your `MONO_PATH`. See the `README` file in each demo directory for a description of the demo.

Targeting Managed Code

By default, Ice for .NET uses unmanaged code for performing protocol compression and for handling signals in the `Ice.Application` class on Windows. You can build a managed version of Ice for .NET that lacks the aforementioned features by editing `config/Make.rules.cs` and uncommenting the `MANAGED=yes` line before you build.

Configuring the bzip2 Library

The `bin` directory contains a file `Ice.dll.config` that configures the bzip2 library. You need to check that the library name specified for the `target` attribute matches the name of the bzip2 library on your machine. If the `target` attribute does not specify the correct name or if bzip2 is not installed, the Ice run time silently runs without protocol compression.

To test whether you have set the attribute correctly, run the server in `demo/Ice/minimal` as follows:

```
$ MONO_LOG_LEVEL=info MONO_LOG_MASK=dll mono --debug server.exe
```

This produces trace output that shows whether the bzip2 library could be located at run time. If the library could be found, you will see a line of trace as follows:

```
Mono-INFO: Found as 'BZ2_bzlibVersion'.
```

Running Mono Applications on Linux

Mono binaries are interpreted so, given a binary called `server.exe`, you have to run the Mono interpreter with an argument `server.exe` to execute that binary:

```
$ mono server.exe
```

If you want to avoid having to explicitly invoke the Mono interpreter, please refer to Registering .exe as non-native binaries.

Installing Ice for .NET on Linux

Open `config/Make.rules.cs` and change the `prefix` variable to hold the top-level installation directory. This directory will be created automatically if necessary. Also review the comments for the `GACINSTALL` variable and decide whether to enable it. Next, run

```
$ make install
```

After installation, add the directory containing the libraries to your `MONO_PATH`. When building applications, you need to reference the libraries with the `-r` option to `mcs`.

Alternatively, you can add the libraries to the global assembly cache. To do this, either set `GACINSTALL` before building the libraries, or use

```
$ gacutil -i <library.dll>
```

Once installed in the cache, the assemblies will always be located correctly without having to set environment variables.

Finally, you could also copy the necessary libraries into the directory that contains the `.exe` for your application.