Building Ice for .NET with Visual Studio

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

On this page:

.NET Build Requirements

Ice for .NET has been extensively tested using the operating systems and compilers listed on our platforms page.

Compiling Ice for .NET with Visual Studio

Unpack the 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 from Ice for C++. 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:

```
> set ICE_HOME=C:\Ice-3.5.1
```

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

```
> cd Ice-3.5.1\cs
```

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

Run nmake:

```
> nmake /f Makefile.mak
```

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

You can also build the demos using the Visual Studio solution located in `demo\demo.sln`.

The demo projects require the Ice Visual Studio Add-In. The add-in is installed automatically when you use the Ice installer, or you can install the add-in manually by following the instructions in `..\vsaddin\INSTALL.txt`.

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:

```
> set ICE_HOME=c:\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 `DEVPATH`. See the `README` file in each demo directory for a description of the demo.

Note that for demos that use IceSSL, the IceSSL plug-in configuration does not contain the fully-qualified name (FQN) for the IceSSL assembly. Instead it just contains the partial name:

```
Ice.Plugin.IceSSL=IceSSL:IceSSL.PluginFactory
```

The IceSSL assembly is found though the use of `DEVPATH`, which is enabled in the `*.exe.config` files. If you want to run an application using the IceSSL assembly installed in the GAC without the use of the `*.exe.config` files, you must add the FQN to the IceSSL plug-in configuration, as shown below:

```
Ice.Plugin.IceSSL=IceSSL, Version=3.5.1.0, Culture=neutral, PublicKeyToken=cdd571ade22f2f16:IceSSL.
PluginFactory
```

Note that `cdd571ade22f2f16` is the token corresponding to ZeroC's public key for signing the assemblies in binary distributions. If you built Ice from sources, your assemblies were signed using the development key instead, which you can find in `config/IceDevKey.snk`. The token for the development key is `1f998c50fec78381`.

SSL Notes for .NET Tests and Demos

In order to use SSL with the tests and sample programs, an SSL certificate must be installed on your system. The configuration files handle this for you, but you will be presented with a confirmation dialog the first time you run a test or sample program.

Once you are finished with the tests and sample programs, follow these steps to remove the certificate:

1. Start Internet Explorer.
2. Select Internet Options from the Tools menu.
3. Select the Content tab and click the "Certificates" button.
4. Select the Trusted Root Certification Authorities tab.
5. Select the entry for "ZeroC Test CA", click the Remove button, and confirm that you want to remove this certificate.

Note that under Windows Vista, the IceSSL configuration test is disabled due to an apparent bug with the .NET framework in which obsolete SSL session ids are reused and cause server authentication to fail. This bug manifests itself in those applications that initialize and destroy multiple IceSSL plug-ins in the same process. As this is an unusual use case, we do not believe it will affect most Ice applications.

Protocol Compression with .NET

Ice for .NET attempts to dynamically load `bzip2.dll` to support protocol compression, therefore this DLL must be present in your `PATH`. Ice automatically disables protocol compression if the DLL cannot be found.

On 64-bit Windows, you must ensure that Ice finds the 64-bit version of `bzip2.dll` before the 32-bit version. The 64-bit and 32-bit bzip2 libraries are installed in *`prefix`*`\bin\x64` and *`prefix`*`\bin`, respectively. For 64-bit Windows, the *`prefix`*`\bin\x64` directory must appear before *`prefix`*`\bir` in your application's `PATH`. (The Ice run time prints a warning to the console if it detects a `bzip2.dll` format mismatch during start-up.)

Installing Ice for .NET on Windows

Run `nmake /f Makefile.mak install` to install Ice for .NET in the directory specified by the `prefix` variable in `config\Make.rules.mak.cs`. After installation, the *`prefix`*`\bin` directory contains executables (such as `iceboxnet.exe`), and the *`prefix`*`\Assemblies` directory contains the .NET assemblies.

GAC Installation

You can add the assemblies to the Global Assembly Cache (GAC). To do this, open Windows Explorer and navigate to the directory `C:\WINDOWS\assembly`. Next, drag and drop (or copy and paste) the assemblies from `Ice-3.5.1\Assemblies` into the right-hand pane to install them in the cache.

You can also use `gacutil` from the command line to achieve the same result:

```
> gacutil /i library.dll
```

The gacutil tool is included with your Visual C# installation. For example, if you have installed Visual Studio 2010 in `C:\Program Files`, the path to gacutil is

```
C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\gacutil.exe
```

Once installed in the cache, the assemblies will always be located correctly without having to set environment variables or copy them into the same directory as an executable.

If you want line numbers for stack traces, you must also install the PDB (`.pdb`) files in the GAC. Unfortunately, you cannot do this using Explorer, so you have to do it from the command line. Open a command shell window and navigate to `C:\WINDOWS\assembly\Microsoft.NET\GAC_MSIL\Ice` (assuming `C:\WINDOWS` is your system root). Doing a directory listing there, you will find a directory named `v4.0_3.5.0.0__`*`UUID`*, for example:

```
v4.0_3.5.0.0__cdd571ade22f2f16
```

Change to that directory (making sure that you use the correct version number for this release of Ice). In this directory, you will see the `Ice.dll` you installed into the GAC in the preceding step. Now copy the `Ice.pdb` file into this directory:

```
> copy path_to_ice.pdb .
```

Targeting Managed Code

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

Targeting .NET Compact Framework

.NET Compact Framework Build Requirements

Ice for .NET Compact Framework requires Microsoft Visual Studio 2008 SP1 and the .NET Compact Framework Version 3.5 SP1.

The default configuration in `config\Make.rules.mak.cs` produces Ice assemblies for use with .NET. If you wish to use Ice with the .NET Compact Framework, you must re-build the sources.

Building Ice for .NET Compact Framework

To build Ice for .NET Compact Framework, open `config\Make.rules.mak.cs` and set `COMPACT=yes`. Run nmake as before:

```
> nmake /f Makefile.mak
```

The .NET Compact Framework assemblies and binaries are generated into the `Assemblies\cf` and `bin\cf` directories, respectively.

Running the .NET Compact Framework Tests

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

```
> python allTests.py --compact
```

Building the .NET Compact Framework Demos

To build the demos, you must first install the Ice Visual Studio Add-In. You can install the Add-in using the Windows MSI installer; for manual installation instructions, refer to the file `..\vsaddin\INSTALL.txt`.

Use the Visual Studio solution located in `demo\democf.sln` to build the demos.

Targeting Silverlight

Silverlight Build Requirements

Ice for Silverlight can be built with the following build configurations:

- Microsoft Visual Studio 2010 SP1 with Silverlight 5.0. There are two additional prerequisites for this build configuration:
  - Microsoft Silverlight 5.0 SDK
  - Microsoft Silverlight 5 Tools
- Microsoft Visual Studio 2012 with Silverlight 5.1 on Windows 8.0.
- Microsoft Visual Studio 2013 with Silverlight 5.1 on Windows 8.1.

The default configuration in `config\Make.rules.mak.cs` produces Ice assemblies for use with .NET. If you wish to use Ice with Silverlight, you must re-build the sources.

Building Ice for Silverlight

To build Ice for Silverlight, open `config\Make.rules.mak.cs` and set `SILVERLIGHT=yes`. Run nmake as before:

```
> nmake /f Makefile.mak
```

The Silverlight assemblies are generated into the `Assemblies\sl` directory.

Installing Ice for Silverlight

To install Ice for Silverlight run

```
> nmake /f Makefile.mak install
```

This will install Ice for Silverlight in the directory specified by the `prefix` variable in `config\Make.rules.mak.cs`. The Ice assemblies will be located in the *`prefix`*`\Assemblies\sl` directory.

Running the Silverlight Tests and Demos

To build the tests and demos, you must first install the Ice Visual Studio Add-In. You can install the Add-in using the Windows MSI installer; for manual installation instructions, refer to the file `..\vsaddin\INSTALL.txt`.

Use the Visual Studio solutions located in `test\testsl.sln` and `demo\demosl.sln` to build the tests and demos, respectively.

Silverlight has no server-side support; you must build the servers from .NET:

```
> cd Ice-3.5.1\cs
> nmake /f Makefile.mak SILVERLIGHT=no
```

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

```
> python allTests.py --silverlight
```

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

To run the demos, open the solution `demo\demosl.sln` and follow the instructions in the `README.txt` file of each demo.

Targeting Unity

Ice for .NET can also be compiled to target the Unity 3 API. To build Ice for the Unity API, open `config\Make.rules.mak.cs` and set `UNITY=yes` . Run `nmake` as before:

```
> nmake /f Makefile.mak
```