.NET Compact Framework Support

Ice for .NET includes support for the .NET Compact Framework (.NET CF).

On this page:

Using Ice for .NET CF

There are several API differences between .NET and .NET CF that impact the Ice run time, therefore Ice for .NET must be re-compiled to target .NET CF. The Ice installer for Windows includes the .NET CF version of the Ice run time in *`install-dir`*`\bin\cf\Ice.dll`, and the Ice Visual Studio Add-in automatically uses this DLL for Smart Device projects. To build Ice for .NET CF in a source distribution, enable `COMPACT` in `cs\config\Make.rules.mak.cs`.

Limitations of Ice for .NET CF

The following features are **not** supported in Ice for .NET CF:

- Protocol compression
- Signal processing in the `Ice.Application` class
- IceSSL
- `ICE_CONFIG` environment variable
- Dynamic loading of Slice checksums
- `Ice.TCP.SndSize` and `Ice.TCP.RcvSize` properties
- Automatic discovery of dependent assemblies containing Slice-generated classes and exceptions

As we discuss in the next section, the last limitation is the most significant.

Managing Factory Assemblies in Ice for .NET CF

When receiving a Slice user exception or a concrete Slice object-by-value, the Ice run time must be able to dynamically translate the encoded Slice type ID (such as `::MyModule::MyType`) into a .NET class name (such as `MyModule.MyType`), dynamically locate that class, and instantiate it. This is convenient for .NET applications because it requires no additional user configuration; at startup, the Ice for .NET run time recursively loads all dependent assemblies used by the program to ensure that any generated classes are available if necessary.

The Compact Framework does not allow a program to discover its dependent assemblies, so this strategy cannot work. Consequently, Ice for .NET CF adds the new configuration property `Ice.FactoryAssemblies` so that you can explicitly list any assemblies that contain the generated code for user exceptions or concrete classes. When searching for a class, Ice for .NET CF first checks in the assemblies specified by this property. If the type is not found, Ice automatically looks in the standard Ice assemblies (`Ice`, `Glacier2`, `IceBox`, `IceGrid`, `IcePatch2`, and `IceStorm`).

Note that the program itself is also considered an assembly. If you compiled the main program directly with Slice-generated code, your `Ice.FactoryAssemblies` property must include the program itself if the generated code includes user exceptions or concrete classes. For simple build scenarios in which all generated code is compiled directly into the executable, the following configuration setting is sufficient:

```
Ice.FactoryAssemblies=client
```

This example assumes the executable is named `client.exe`. On the other hand, if Slice-generated code is also compiled into a dependent assembly, your configuration might look like this instead:

```
Ice.FactoryAssemblies=client MyOtherAssembly
```

Failing to define `Ice.FactoryAssemblies` can cause the Ice run time in the receiver to raise `NoObjectFactoryException` or `UnmarshalOutOfBoundsException`. If you are experiencing either of these exceptions, verify that your assemblies are configured correctly.

See Also

- Visual Studio Add-in
- `Ice.FactoryAssemblies`