Backward Compatibility of Ice Versions

The subsections below describe how Ice maintains (or does not maintain) backwards compatibility between versions.

On this page:

Source-code compatibility

Ice maintains source-code compatibility between a patch release (e.g., 3.7.1) and the most recent minor release (e.g., 3.7.0), but does not guarantee source-code compatibility between minor releases (e.g., between 3.6 and 3.7).

Upgrading your Application from Ice 3.6 describes the significant API changes in this release that may impact source-code compatibility. Furthermore, the subsections Removed APIs and Deprecated APIs summarize additional changes to Ice APIs that could affect your application.

Back to Top ^

Binary compatibility

As for source-code compatibility, Ice maintains backward binary compatibility between a patch release and the most recent minor release, but does not guarantee binary compatibility between minor releases.

The requirements for upgrading to a new minor (or major) release depend on the language mapping used by your application:

- For statically-typed languages (C++, Java, .NET), the application must be recompiled.
- For scripting languages that use static translation, your Slice files must be recompiled.
- No action is necessary for a Python or Ruby script that loads its Slice files dynamically.

Back to Top ^

On-the-wire compatibility

Ice always maintains "on the wire" compatibility with prior releases. A client using Ice version *x* can communicate with a server using Ice version *y* and vice versa.

The Ice message format is governed by two sets of rules: the protocol rules and the encoding rules. The protocol rules define the number of message types as well as the format of the message headers. The encoding rules specify how Slice types are marshaled. Ice maintains separate versions for the protocol and encoding, which makes it possible for them to evolve independently.

Several features introduced in Ice 3.5 required changes to the way that Slice classes and exceptions are marshaled. These changes make the encoding incompatible with previous releases, and therefore Ice 3.5 introduced version 1.1 of the Ice encoding. No additional encoding changes were made in Ice 3.6 and Ice 3.7.

The protocol remains the same since the first version Ice, with protocol version number 1.0.

Changing the protocol or encoding format can be disruptive and does not happen often. Naturally, these changes raise the issue of backward compatibility with applications that use earlier versions of Ice, and more specifically, applications that use version 1.0 of the Ice encoding. Ice 3.5, 3.6 and 3.7 use the following semantics:

- Encoding version 1.1 is the default for these releases. You can change this default using the `Ice.Default.EncodingVersion` property.
- A proxy is marshaled with an embedded encoding version representing the highest version supported by the target object, so the receiver of a proxy always knows the encoding version(s) it is allowed to use when invoking operations on that proxy. A proxy created locally uses the default encoding version, which can be inspected and changed using proxy methods.
- The results of an operation are encoded using the same version as the incoming request.
- As you would expect, the features provided by the 1.1 encoding are unavailable when an Ice application is forced to use the 1.0 encoding.

See Encoding Version 1.1 for additional details.

Back to Top ^

Interface compatibility

Although Ice always maintains compatibility at the protocol level, changing Slice definitions can also lead to incompatibilities. As a result, Ice maintains interface compatibility between a patch release and the most recent minor release, but does not guarantee compatibility between minor releases.

This issue is particularly relevant if your application uses Ice services such as IceGrid or IceStorm, as a change to an interface in one of these services may adversely affect your application.

Interface changes in an Ice service can also impact compatibility with its administrative tools, which means it may not be possible to administer an Ice 3.7 service using a tool from a previous minor release (or vice-versa).

Back to Top ^

IceGrid

Starting with Ice 3.2, IceGrid registries and nodes are interface-compatible so you can use different IceGrid registry and node versions within the same deployment. For example, you can use an IceGrid node from Ice 3.2 with a registry from Ice 3.7.

IceGrid master and slaves can have different versions as long as they use the same database format. Since the IceGrid database format changed in Ice 3.6, an IceGrid slave from earlier versions will not be able to synchronize with an IceGrid master from Ice 3.6 or later. You need to upgrade all your IceGrid registries to Ice 3.6 or 3.7 to ensure replication between the registries work.

An IceGrid node can activate Ice servers that use different Ice releases, for example, an IceGrid node version 3.7 can activate an Ice 3.4 server.

The IceGrid graphical and command-line administrative tools can only administer an IceGrid registry with the same `major.minor` version. For example, you have to use the 3.7 administrative tools to administer an IceGrid 3.7 registry; you cannot use an administrative tool from an earlier Ice version. The reverse is also true: you cannot administer an IceGrid 3.6 registry with an IceGrid 3.7 administration tool.

Back to Top ^

IceStorm

Topic linking is supported between all IceStorm versions released after 3.0.

Back to Top ^