Upgrading your Application from Ice 3.4

The subsections below provide additional information about upgrading to Ice 3.5, including administrative procedures for the supported platforms.

On this page:

Backward compatibility of Ice versions

A discussion of backward compatibility in Ice involves many factors.

Source-code compatibility

Ice maintains source-code compatibility between a patch release (e.g., 3.4.2) and the most recent minor release (e.g., 3.4.0), but does not guarantee source-code compatibility between minor releases (e.g., between 3.4 and 3.5).

The subsections below describe the significant API changes in this release that may impact source-code compatibility. Furthermore, the subsections Removed APIs in Ice 3.5 and Deprecated APIs in Ice 3.5 summarize additional changes to Ice APIs that could affect your application.

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 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.

On-the-wire compatibility

Ice always maintains protocol ("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.

Several features introduced in Ice 3.5 require a new version of the Ice encoding, encoding version 1.1. Older versions of Ice do not understand this encoding: you need to use Ice encoding version 1.0 for communications between clients or servers using Ice 3.5 and clients and servers using older Ice versions. See Encoding Version 1.1 for details.

Database compatibility

Upgrading to a new minor release of Ice often includes an upgrade to the supported version of Berkeley DB. In turn, this may require an application to migrate its databases, either because the format of Berkeley DB's database files has changed, or due to a change in the schema of the data stored in those databases.

For example, if your application uses Freeze, it may be necessary for you to migrate your databases even if your schema has not changed.

Certain Ice services also use Freeze in their implementation. If your application uses these services (IceGrid and IceStorm), it may be necessary for you to migrate their databases as well.

Please refer to the relevant subsections below for migration instructions.

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 3.5.x service using a tool from a previous minor release (or vice-versa).

IceGrid

Starting with Ice 3.2.0, IceGrid registries and nodes are interface-compatible. For example, you can use an IceGrid node from Ice 3.2 with a registry from Ice 3.4.

IceGrid registry replication is only supported between registries using Ice 3.3 or later. With Ice 3.5, since the database format changed, an IceGrid slave from Ice 3.4 or 3.3 will not be able to synchronize with an IceGrid master from Ice 3.5. You need to upgrade all your IceGrid registries to Ice 3.5 to ensure replication between the registries work.

An IceGrid node using Ice 3.5 or later is able to activate a server that uses Ice < 3.5.

Regarding the IceGrid graphical and command-line administrative tools, you must use the 3.5 administrative tools to administer an IceGrid 3.5 registry; you cannot use an administrative tool from an earlier Ice version. The reverse is also true: you cannot administer an IceGrid 3.4 registry with an IceGrid 3.5 administration tool.

IceStorm

Topic linking is supported between all IceStorm versions released after 3.0.0.

Slicing behavior in Ice 3.5

In version 1.0 of the Ice encoding, Slice classes and user exceptions are always marshaled in a format that allows a receiver to "slice" an unknown derived type to a known base type. An application that uses Ice 3.4 or earlier may depend on this behavior. For example, the slicing feature makes it possible for a sender to evolve independently of a receiver, adding more derived types that the receiver does not understand without causing the receiver to fail should it encounter one of these new types.

Version 1.1 of the Ice encoding introduces two formats for classes and exceptions. The compact format, which is used by default, does not support the slicing feature, meaning an application that relies on this feature may begin to receive `NoObjectFactoryException` or `UnknownUserExcepti` on errors after upgrading to Ice 3.5. There are several possible solutions:

- Continue using version 1.0 of the Ice encoding by setting `Ice.Default.EncodingVersion` to `1.0` in all senders and receivers.
- If you wish to take advantage of other features offered by encoding version 1.1, you can make the sliced format the default by setting `Ice.Default.SlicedFormat` to 1 in all senders and receivers.
- Annotate your interfaces or operations with metadata so that the sliced format is only used where necessary.

Slice language changes in Ice 3.5

The term `optional` is now a Slice keyword in Ice 3.5, which means existing Slice definitions that use this term as an identifier will generate an error. To continue using `optional` as an identifier, you can use the Slice syntax for escaping keywords:

Slice

```
struct Example
{
    bool \optional;
};
```

C++ changes in Ice 3.5

The `slice2cpp` translator no longer generates C++ comparison operators for all Slice structures. Comparison operators are only generated for structures that qualify as a legal dictionary key type. To generate a comparison operator for a Slice structure that doesn't qualify as such, you can prefix the `["cpp:comparable"]` metadata to your Slice struct definition.

Java changes in Ice 3.5

The generated code and supporting classes for the various Ice services are now provided as separate JAR files:

JAR File	Description
`Freeze.jar`	Freeze classes
`Glacier2.jar`	Glacier2 generated code and helper classes
`Ice.jar`	The core Ice run time, including the IceSSL plugin
`IceBox.jar`	IceBox server, generated code, admin utility
`IceGrid.jar`	IceGrid generated code
`IcePatch2.jar`	IcePatch2 generated code
`IceStorm.jar`	IceStorm generated code

Also note that the manifest in `Freeze.jar` no longer contains a reference to `db.jar`, so you may need to update your class path to include `db.jar`.

The Eclipse Plug-in still includes only `Ice.jar` by default, but you can easily include additional JAR files by changing the Slice2Java properties for your project.

Python changes in Ice 3.5

Ice 3.5 adds support for Python 3. Developers who are migrating existing Ice applications from Python 2 to Python 3 should be aware of the following changes that affect the Python language mapping:

- One of the most significant changes in Python 3 is its unification of the unicode and string types into just one type, `str`. Python developers must convert any existing unicode literals and variables into their string equivalents. For Ice developers, a benefit of this unification is that all string values sent "over the wire" are now sent and received as unicode strings. With Python 2.x, the Ice run time behavior remains unchanged: string and unicode values are both accepted as inputs, but only string values are returned as outputs.
- The Python 3 mapping for the Slice type `sequence<byte>` now defaults to the new Python type `bytes`. A string value is no longer accepted as a legal value for `sequence<byte>` with Python 3.

PHP changes in Ice 3.5

For compatibility with newer PHP releases, the Ice extension no longer supports "call-time" reference arguments when calling `Ice_initialize` and `Ice_createProperties`. For example, existing applications may invoke `Ice_initialize` with an explicit reference argument as follows:

PHP

```
$communicator = Ice_initialize(&$argv);
```

The proper way to call this function using Ice 3.5 is shown below:

PHP

```
$communicator = Ice_initialize($argv);
```

Ice-specific arguments are now removed from the given argument vector.

IPv6 changes in Ice 3.5

IPv6 is now enabled by default on all platforms. You can disable the use of IPv6 in Ice by setting the `Ice.IPv6` property to `0`, which may be necessary in environments that lack IPv6 support or on older platforms such as Windows XP. The new property `Ice.PreferIPv6Address` determines whether IPv4 or IPv6 addresses take precedence when resolving host names.

Migrating IceGrid databases from Ice 3.4

Ice 3.5 supports the migration of IceGrid databases from Ice 3.3 and from Ice 3.4. To migrate from earlier Ice versions, you will first need to migrate the databases to the Ice 3.3 format. If you require assistance with such migration, please contact support@zeroc.com.

To migrate, first stop the IceGrid registry you wish to upgrade.

Next, copy the IceGrid database environment to a second location:

```
$ cp -r db recovered.db
```

Locate the correct version of the Berkeley DB recovery tool (usually named `db_recover`). It is essential that you use the `db_recover` executable that matches the Berkeley DB version of your existing Ice release. For Ice 3.3, use `db_recover` from Berkeley DB 4.6. For Ice 3.4, use `db_recover` from Berkeley DB 4.8. You can verify the version of your `db_recover` tool by running it with the `-V` option:

```
$ db_recover -V
```

Now run the utility on your copy of the database environment:

```
$ db_recover -h recovered.db
```

Change to the location where you will store the database environments for IceGrid 3.5:

```
$ cd <new-location>
```

Next, run the `upgradeicegrid35.py` utility located in the `config` directory of your Ice distribution (or in `/usr/share/Ice-3.5` if using an RPM installation). The first argument is the path to the old database environment. The second argument is the path to the new database environment.

In this example we'll create a new directory `db` in which to store the migrated database environment:

```
$ mkdir db
$ upgradeicegrid35.py <path-to-recovered.db> db
```

Upon completion, the `db` directory contains the migrated IceGrid databases.

By default, the migration utility assumes that the servers deployed with IceGrid also use Ice 3.5. If your servers still use an older Ice version, you need to specify the `--server-version` command-line option when running `upgradeicegrid35.py`:

```
$ upgradeicegrid.py --server-version 3.4.2 <path-to-recovered.db> db
```

The migration utility will set the `server` descriptor attribute `ice-version` to the specified version and the IceGrid registry will generate configuration files compatible with the given version.

If you are upgrading the master IceGrid registry in a replicated environment and the slaves are still running, you should first restart the master registry in read-only mode using the `--readonly` option, for example:

```
$ icegridregistry --Ice.Config=config.master --readonly
```

Next, you can connect to the master registry with `icegridadmin` or the IceGrid administrative GUI from Ice 3.5 to ensure that the database is correct. If everything looks fine, you can shutdown and restart the master registry without the `--readonly` option.

IceGrid slaves from Ice 3.3 or 3.4 won't interoperate with the IceGrid 3.5 master. You can leave them running during the upgrade of the master to not interrupt your applications. Once the master upgrade is done, you should upgrade the IceGrid slaves to Ice 3.5 using the instruction above.

Migrating IceStorm databases from Ice 3.4

No changes were made to the database schema for IceStorm in this release. However, you still need to update your databases as described below.

Migrating Freeze databases from Ice 3.4

No changes were made that would affect the content of your Freeze databases. However, we upgraded the version of Berkeley DB and the new 1.1 encoding may require that you re-create your indices or set some additional configuration if you use Freeze maps. If you only use Freeze evictors, you only need to upgrade your databases to the new Berkeley DB format. See Encoding Version 1.1.

Upgrading to the Berkeley DB 5.3 format

When upgrading to Ice 3.5, you must upgrade your database to the Berkeley DB 5.3 format. The only change that affects Freeze is the format of Berkeley DB's log file.

The instructions below assume that the database environment to be upgraded resides in a directory named `db` in the current working directory. For a more detailed discussion of database migration, please refer to the Berkeley DB Upgrade Process.

To migrate your database:

1. Shut down the old version of the application.
2. Make a backup copy of the database environment:

```
> cp -r db backup.db      (Unix)
> xcopy /E db backup.db   (Windows)
```

3. Locate the correct version of the Berkeley DB recovery tool (usually named `db_recover`). It is essential that you use the `db_recover` executable that matches the Berkeley DB version of your existing Ice release. For Ice 3.4, use `db_recover` from Berkeley DB 4.8. You can verify the version of your `db_recover` tool by running it with the `-V` option:

```
> db_recover -V
```

4. Use the `db_recover` tool to run recovery on the database environment:

```
> db_recover -h db
```

5. Recompile and install the new version of the application.
6. Force a checkpoint using the `db_checkpoint` utility. Note that you must use the `db_checkpoint` utility from Berkeley DB 5.3 when performing this step.

```
> db_checkpoint -1 -h db
```

7. Restart the application.

Freeze maps and the 1.1 encoding

The majority of Freeze maps will work out of the box with the new 1.1 encoding and without additional configuration. However there are a few cases where you will be required to take special action or add some configuration:

- If your Freeze maps use keys with Slice enumerations having more than 127 elements, the encoding of those keys will be incompatible with the 1.1 encoding. You will need to set the `Freeze.DbEnv.<env-name>.EncodingVersion` property to `1.0` to ensure that you decode the keys with the 1.0 encoding and not with the default 1.1 encoding.
- If your Freeze maps use indices and those indices use Slice classes, or use Slice enumerations having more than 127 elements, you have two options:
    1. Re-create the indices with the 1.1 encoding. To recreate your indices, you have to recreate the Freeze maps using the `recreate` method generated on the Freeze map classes by `slice2freezej`. See the Freeze Maps documentation for more information.
    2. Set the `Freeze.DbEnv.<env-name>.EncodingVersion` property to `1.0` to ensure that Freeze still uses the 1.0 encoding for your indices.
- If you use indices with Java Freeze maps and the map value uses classes, you have to recreate the indices because of a bug in the encoding of the Ice 3.4 indices. This only affects Java Freeze maps; C++ Freeze maps are not affected by this issue.

If you configure your Freeze database environment to use the 1.0 encoding, you will not be able to take advantage of the new features provided with the 1.1 encoding, such as optional values and the new formats for classes.

Migrating Android applications from Ice 3.4

Prior versions of the Ice plug-in for Eclipse (including the version released for Ice 3.5b) created a workspace variable named `ICE_HOME` that referred to the Ice installation directory configured in Eclipse's *Preferences* dialog. After adding the Slice2Java builder to a project, the builder updated the project's build path to include Ice JAR files relative to `ICE_HOME`. For example, `Ice.jar` was configured as the library reference `ICE_HOME/lib/Ice.jar`.

As of Ice 3.5.0, the Eclipse plug-in no longer uses the `ICE_HOME` variable. Instead, it now uses the workspace variable `ICE_JAR_HOME` to refer to the subdirectory containing `Ice.jar`, equivalent to the previous setting of `ICE_HOME/lib`. Consequently, a new project's build path now includes the library reference `ICE_JAR_HOME/Ice.jar`.

The plug-in does not attempt to modify the build path of an existing Ice for Android project to use `ICE_JAR_HOME`, nor does it remove `ICE_HOME` from the workspace. An existing project can continue to use the `ICE_HOME` variable in its build path, but moving or copying the project to a new workspace where `ICE_HOME` is no longer defined means the project's build path must be updated.

Changed APIs in Ice 3.5

This section describes APIs whose semantics have changed, potentially in ways that are incompatible with previous releases.

The following APIs were changed in Ice 3.5:

- The dynamic invocation and dispatch API now requires encapsulations for input parameters and output results. For example, when calling `ice_invoke`, the value for `inParams` must be an encapsulation of the input parameters:

  C++

  ```
  Ice::OutputStreamPtr out = Ice::createOutputStream();
  out->startEncapsulation();
  // marshal parameters
  out->endEncapsulation();
  Ice::ByteSeq inParams;
  out->finished(inParams);
  // invoke operation...
  ```

  Similarly, the byte sequence containing the results must also be an encapsulation:

  C++

  ```
  Ice::ByteSeq results;
  // invoke operation...
  Ice::InputStreamPtr in = Ice::wrapInputStream(results);
  in->startEncapsulation();
  // unmarshal results
  in->endEncapsulation();
  ```

- The streaming API uses new methods to mark the beginning and end of objects and exceptions: `startObject`/`endObject` and `startException`/`endException`.

Removed APIs in Ice 3.5

This section describes APIs that were deprecated in a previous release and have now been removed. Your application may no longer compile successfully if it relies on one of these APIs.

The following APIs were removed in Ice 3.5:

- `Ice.Default.CollocationOptimization`
  Use `Ice.Default.CollocationOptimized` instead.

- *`proxy`*`.CollocationOptimization`
  Use *`proxy`*`.CollocationOptimized` instead.

- *`adapter`*`.RegisterProcess`
  This property caused the Ice run time to register a proxy with the locator registry that allowed the process to be shut down remotely. The new administrative facility has replaced this functionality.

- `Ice.ServerId`
  As with *`adapter`*`.RegisterProcess`, this property was used primarily for IceGrid integration and has been replaced by a similar mechanism in the administrative facility.

- `Ice.Trace.Location`
  This property has been replaced by `Ice.Trace.Locator`.

- `Glacier2.Admin` and `IcePatch2.Admin`
  These are the names of administrative object adapters in Glacier2 and IcePatch2, respectively. The functionality offered by these object adapters has been replaced by that of the administrative facility, therefore these adapters (and their associated configuration properties) are no longer necessary.

- `Ice.Util.generateUUID()`
  In Java use `java.util.UUID.randomUUID().toString()`. In C# use `System.Guid.NewGuid.ToString()`.

- `Object.ice_hash()`
  `ObjectPrx.ice_getHash()`
  `ObjectPrx.ice_toString()`
  Ice no longer defines hash methods for objects and proxies. To convert a proxy into a string, use the standard platform methods: `toString` in Java, `ToString` in C#.

Deprecated APIs in Ice 3.5

This section discusses APIs and components that are now deprecated. These APIs will be removed in a future Ice release, therefore we encourage you to update your applications and eliminate the use of these APIs as soon as possible.

The following APIs were deprecated in Ice 3.5:

- `Stats` facility
  This functionality is now provided by the Instrumentation facility and the Metrics facet.

The following components were deprecated in Ice 3.5:

- Qt SQL database plug-ins for IceGrid and IceStorm
  Freeze is now the only supported persistence mechanism for these services.