Upgrading your Application from Ice 3.2 or Earlier Releases

In addition to the information provided in Upgrading your Application from Ice 3.3, users who are upgrading from Ice 3.2 or earlier should also review this page.

On this page:

Migrating IceStorm databases from Ice 3.2

Ice 3.4 supports the migration of IceStorm databases from Ice 3.1 and from Ice 3.2. Migration from earlier Ice versions may work, but is not officially supported. If you require assistance with such migration, please contact support@zeroc.com.

To migrate, first stop your IceStorm servers.

Next, copy the IceStorm 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 3.1, use `db_recover` from Berkeley DB 4.3.29. For Ice 3.2, use `db_recover` from Berkeley DB 4.5. 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 IceStorm 3.4:

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

Next, run the `icestormmigrate` utility. 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
$ icestormmigrate <path-to-recovered.db> db
```

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

Back to Top ^

Migrating IceGrid databases from Ice 3.2

Ice 3.4 supports the migration of IceGrid databases from Ice 3.1 and from Ice 3.2. Migration from earlier Ice versions may work, but is not officially supported. 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 3.1, use `db_recover` from Berkeley DB 4.3.29. For Ice 3.2, use `db_recover` from Berkeley DB 4.5. 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 3.4:

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

Next, run the `upgradeicegrid.py` utility located in the `config` directory of your Ice distribution (or in `/usr/share/Ice-3.4.1` 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
$ upgradeicegrid.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 3.4. If your servers still use an older Ice version, you need to specify the `--server-version` command-line option when running `upgradeicegrid.py`:

```
$ upgradeicegrid.py --server-version 3.2.1 <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 to ensure that the database is correct. If everything looks fine, you can shutdown and restart the master registry without the `--readonly` option.

Back to Top ^

Migrating Freeze databases from Ice 3.2

No changes were made that would affect the content of your Freeze databases. However, we upgraded the version of Berkeley DB, therefore when upgrading to Ice 3.4 you must also upgrade your database to the Berkeley DB 4.8 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 3.1, use `db_recover` from Berkeley DB 4.3.29. For Ice 3.2, use `db_recover` from Berkeley DB 4.5. 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 4.8 when performing this step.

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

7. Restart the application.

Back to Top ^

Removed APIs in Ice 3.3

This section describes APIs that were deprecated in a previous release and have been removed in Ice 3.3. Your application may no longer compile successfully if it relies on one of these APIs.

Please refer to Removed APIs in Ice 3.4.0 for information on APIs that were removed in Ice 3.4.

Back to Top ^

Thread per connection

The primary purpose of this concurrency model was to serialize the requests received over a connection, either because the application needed to ensure that requests are dispatched in the order they are received, or because the application did not want to implement the synchronization that might be required when using the thread pool concurrency model.

Another reason for using the thread-per-connection concurrency model is that it was required by the IceSSL plug-ins for Java and C#. This requirement has been eliminated.

The ability to serialize requests is now provided by the thread pool and enabled via a new configuration property:

`<threadpool>.Serialize=1`

Please refer to the manual for more details on this feature.

Aside from the potential semantic changes involved in migrating your application to the thread pool concurrency model, other artifacts of thread-per-connection may be present in your application and must be removed:

- The configuration properties `Ice.ThreadPerConnection` and `<proxy>.ThreadPerConnection`
- The proxy methods `ice_threadPerConnection` and `ice_isThreadPerConnection`

Back to Top ^

.NET metadata

The metadata directive `cs:collection` is no longer valid. Use `["clr:collection"]` instead.

Back to Top ^

C++

The following C++ methods have been removed:

- `Application::main(int, char*[], const char*, const Ice::LoggerPtr&)`

  Use `Application::main(int, char*[], const InitializationData&)` instead.

- `initializeWithLogger`
- `initializeWithProperties`
- `initializeWithPropertiesAndLogger`

  Use `initialize(int, char*[], const InitializationData&)` instead.

- `stringToIdentity`
- `identityToString`

  Use the equivalent operations on `Communicator`.

Back to Top ^

Java

The following methods have been removed:

- `Application.main(String, String[], String, Logger)`

  Use `Application.main(String, String[], InitializationData)` instead.

- `initializeWithLogger`
- `initializeWithProperties`
- `initializeWithPropertiesAndLogger`

  Use `initialize(String[], InitializationData)` instead.

Back to Top ^

.NET

The following methods have been removed:

- `Application.main(string, string[], string, Logger)`

  Use `Application.main(string, string[], InitializationData)` instead.

- `initializeWithLogger`
- `initializeWithProperties`
- `initializeWithPropertiesAndLogger`

  Use `initialize(ref string[], InitializationData)` instead.

Back to Top ^

Python

The following methods have been removed:

- `initializeWithLogger`
- `initializeWithProperties`
- `initializeWithPropertiesAndLogger`

  Use `initialize(args, initializationData)` instead.

- `stringToIdentity`
- `identityToString`
  Use the equivalent operations on `Communicator`.

Back to Top ^

General

The following methods have been removed:

- `ice_hash`
- `ice_communicator`
- `ice_collocationOptimization`

- `ice_connection`
  These proxy methods were replaced by ones of the form `ice_get...`, such as `ice_getCommunicator`. `ice_collocationOptimization` is now `ice_getCollocationOptimized`.

- `ice_newIdentity`
- `ice_newContext`
- `ice_newFacet`
- `ice_newAdapterId`
- `ice_newEndpoints`
  These proxy methods were replaced by ones that do not use `new` in their names. For example, `ice_newIdentity` was replaced by `ice_identity`.

Back to Top ^

`Ice.LoggerPlugin`

This property provided a way to install a custom logger implementation. It has been replaced by a more generalized facility for installing custom loggers.

Back to Top ^

Deprecated APIs in Ice 3.3

This section discusses APIs and components that are deprecated in Ice 3.3. 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.

Please refer to Deprecated APIs in Ice 3.4.0 for information on APIs that were deprecated in Ice 3.4.

Back to Top ^

Sequences as dictionary keys

The use of sequences, and structures containing sequences, as the key type of a Slice dictionary is now deprecated.

Back to Top ^

`LocalObject`

The mappings for the `LocalObject` type have changed in Java, .NET, and Python. The new mappings are shown below:

Java	`java.lang.Object`
.NET	`System.Object`
Python	`object`

The types `Ice.LocalObject` and `Ice.LocalObjectImpl` are deprecated.

Back to Top ^

`Ice.Trace.Location`

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

Back to Top ^

`Ice.Default.CollocationOptimization`

This property, as well as the corresponding proxy property, have been replaced by `Ice.Default.CollocationOptimized` and `<proxy>.CollocationOptimized`, respectively.

Back to Top ^

`<Adapter>.RegisterProcess`

This property caused the Ice run time to register a proxy with the locator registry (e.g., IceGrid) that allowed the process to be shut down remotely. The new administrative facility has replaced this functionality.

Back to Top ^

`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.

Back to Top ^

`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 deprecated.

Back to Top ^