# Upgrading your Application from Ice 3.6

The subsections below provide additional information about upgrading to Ice 3.7, including administrative procedures for the supported platforms.

On this page:



## C++ Changes

You should be able to rebuild your C++ source code with Ice 3.7 with few if any changes, provided you use the Ice C++98 mapping of Ice 3.7. Upgrading to the new Ice C++11 mapping is a bigger undertaking which requires extensive changes to your source code. As a result, this section deals only with upgrades to Ice 3.7 with the C++98 mapping.

> See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

## IceUtil Library Removed

The core Ice libraries in Ice 3.6 and prior releases were Ice and IceUtil. As of Ice 3.6, IceUtil was merged into Ice, so you can no longer link with IceUtil. On Linux and macOS, you need to remove `-lIceUtil` from your Makefiles. On Windows, you probably don't need to do anything since all Ice libraries are linked through `pragma comment lib` directives in header files (this linking through header files was introduced in Ice 3.6.0).

## Stream API

We made significant changes to the Streaming interfaces in all language mappings. In C++, `InputStream` and `OutputStream` are now created through their constructors and typically stack allocated; in previous releases, they were heap-allocated and created using factory functions in the `Ice` namespace. Please refer to C++ Streaming Interfaces for complete details.

The command-line option `--stream` is no longer supported by `slice2cpp`.

## Dispatch Interceptors

The dispatch and ice_dispatch functions have changed slightly: they now return a bool that indicates whether the request was dispatched synchronously (true) or asynchronously with AMD (false). They previously returned an enumerator.

## IceSSL Certificate Creation

IceSSL Certificates are now created through factory functions, such as `cert = IceSSL::Certificate::load("myCert.pem")`. In previous releases, you would create them directly with the constructors of the `Certificate` class.

## IceSSL Connection Info

The `certs` member of `IceSSL::ConnectionInfo` class is now a sequence of native certificate objects; in previous releases it was a sequence of string elements containing the PEM encoded certificates. The `IceSSL::NativeConnectionInfo` type that used to provide the native certificates has been removed.

## OpenSSL Context with IceSSL

The member functions `setContext` and `getContext`, used to set or retrieve an OpenSSL context, are now on the `IceSSL::OpenSSL::Plugin` class.

## C++11 Extensions

Ice 3.6 provided a few extensions for C++11-capable compilers, primarily additional AMI overloads with `std::function` parameters (suitable for lambda expression arguments). These extensions are no longer included in the C++98 mapping. If you want to take advantage of the C++11 features provided by your C++ compiler, you should upgrade to the Ice C++11 mapping.

## `lib/c++11` and ++11 Libraries on Linux

The Ice 3.6 binary distributions for Linux include libraries with a ++11 suffix, such as `libIce++11.so.36`, and a `c++11` subdirectory in `lib` or `lib64` with symbolic links to these ++11 libraries. These libraries correspond to a build of Ice using GCC with `--std=c++11` turned on. This "C++11 build" of Ice 3.6 provides some C++11 extensions available only in C++11 mode (see above).

In Ice 3.7, these ++11 libraries correspond to the Ice C++11 mapping, and there is no longer a `c++11` subdirectory of `lib`. With Ice 3.7 and GCC 4.8.2 or greater, you should be able to use the Ice C++98 binaries in any mode (`default`, `--std=c++11`, etc.). See the GCC Cxx11AbiCompatibility page for more information on ABI compatibility with GCC. Symbolic links to these C++98 libraries (`libIce.so`, `libIceGrid.so`, etc.) are in the main `lib` directory.

Back to Top ^

# C# Changes

You should be able to rebuild your C# source code with Ice 3.7 with few if any changes.

> See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

## AMD

The AMD (asynchronous method dispatch) mapping has been updated to be Task-based. Applications using AMD should update their servant implementations to use the new Task-based mapping. Refer to the Asynchronous Method Dispatch (AMD) in C-Sharp page for details of the new AMD mapping.

### Stream API

We made significant changes to the Streaming interfaces in all language mappings. In C#, `InputStream` and `OutputStream` are now created through their constructors; in previous releases, they were created using factory functions in the `Ice.Util` class. Please refer to C-Sharp Streaming Interfaces for complete details.

The command-line option `--stream` is no longer supported by `slice2cs`.

### Dispatch Interceptors

The dispatch and ice_dispatch functions have changed slightly: they now return a `Task<Ice.OutputStream>` for asynchronous dispatch or null otherwise. They previously returned an enumerator.

### IceSSL Connection Info

The `certs` member of `IceSSL::ConnectionInfo` class is now a sequence of native certificate objects; in previous releases it was a sequence of string elements containing the PEM encoded certificates. The `IceSSL.NativeConnectionInfo` type that used to provide the native certificates has been removed.

Back to Top ^

## Java Changes

You should be able to rebuild your Java source code with Ice 3.7 with few if any changes, provided you use the Java Compat mapping of Ice 3.7. Upgrading to the new Java mapping is a bigger undertaking which requires extensive changes to your source code. As a result, this section deals only with upgrades to Ice 3.7 with the Java Compat mapping.

> See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

### New `slice2java` Option

To use the Java Compat mapping, you must add the new `--compat` option to your invocations of `slice2java`. For Gradle projects, you can set the property `slice.compat = true` to enable the Java Compat mapping.

### JAR Filenames

The names of the JAR files for the Java Compat run time now include `compat`, such as `ice-compat-3.7.0.jar`.

### `IceUtil` Package Removed

The following classes have moved to the Freeze repository:

- `IceUtil.Cache`
- `IceUtil.FileLockException`
- `IceUtil.Store`

### Stream API

We made significant changes to the Streaming interfaces in all language mappings. In Java, `InputStream` and `OutputStream` are now created through their constructors; in previous releases, they were created using factory functions in the `Ice.Util` class. Please refer to Java Streaming Interfaces for complete details.

The command-line option `--stream` is no longer supported by `slice2java`.

### Dispatch Interceptors

The dispatch and ice_dispatch functions have changed slightly: they now return a bool that indicates whether the request was dispatched synchronously (true) or asynchronously with AMD (false). They previously returned an enumerator.

### IceSSL Connection Info

The `certs` member of `IceSSL::ConnectionInfo` class is now a sequence of native certificate objects; in previous releases it was a sequence of string elements containing the PEM encoded certificates. The `IceSSL.NativeConnectionInfo` type that used to provide the native certificates has been removed.

Back to Top ^

## JavaScript Changes

See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

### Class Helpers

The `Ice.Class` helper method has been removed. The JavaScript `class` keyword or a third-party helper should be used to declare JavaScript classes.

### Dictionary Mapping

The mapping for dictionaries has been updated to use the standard JavaScript `Map` type when possible. `Ice.HashMap` is still used for dictionaries with mutable keys and its API has been updated to match that of JavaScript Map.

### Promise Usage

The `Ice.Promise` class in previous version was a custom implementation of the Promise/A+ specification. It has been updated to be an extension of the standard JavaScript Promise type and most of the non-standard methods have been removed:

- `Ice.Promise.prototype.exception` method has been removed, use Promise.prototype.catch instead.
- `Ice.Promise.prototype.succeed` has been removed, use Promise.prototype.resolve instead. The `succeed` method accepted a variable number of arguments; with `resolve` you can achieve the same by passing an array with the values.
- `Ice.Promise.prototype.fail` has been removed, use Promise.prototype.reject instead. The `fail` method accepted a variable number of arguments; with `reject` you can achieve the same by passing an array with the values.
- `Ice.Promise.prototype.succeeded`, `Ice.Promise.prototype.failed` and `Ice.Promise.prototype.completed` methods have been removed and there are replacements in the standard Promise type. These methods were rarely used in practice.
- `Ice.Promise` completion callbacks no longer provide an `Ice.AsyncResult` parameter as the last argument. If you need to use it you must keep a reference to it when invoking a method.
- `Ice.Promise.all` has been removed, use Promise.all instead.

### AMD

The ["amd"] metadata is ignored by the `slice2js` compiler. The compiler no longer generates a separate method that receives an AMD callback with `ice_response` and `ice_exception` member methods. Instead a method can take advantage of AMD (asynchronous method dispatch) by returning a Promise object from a servant method.

### Mapping for Sequence of Bytes

The mapping for `sequence<byte>` is always the `Uint8Array` JavaScript type; previously the NodeJS engine used a NodeJS `Buffer` type and browser engines used a `Uint8Array`. The helper method `Ice.Buffer.createNative` has been removed; the `Uint8Array` constructor should be used instead.

Back to Top ^

## Objective-C Changes

See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

### Dispatch Interceptors

The dispatch and ice_dispatch functions have changed slightly: they no longer return a bool, user exceptions are now raised by `ice_dispatch`.

### ICEInputStream

The `wrapInputStream` method has been removed from the `ICEUtil` class.

Back to Top ^

## PHP changes

See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

### Namespace Usage

The Ice for PHP extension included in binary distributions is now built with namespaces enabled. All Ice definitions are placed inside Ice namespaces, and the default mapping for a Slice module is a PHP namespace with the same name. The old flattened mapping has been deprecated and will be removed in a future release.

To use the old flattened mapping, you need a custom build of the Ice for PHP extension with namespaces disabled and you need to pass the `--no-namespaces` option to `slice2php` when compiling your Slice files. Consult the PHP build instructions for details of how to build the PHP extension with namespaces disabled.

### Loading Ice

The Ice run time is loaded by `require Ice.php` independently of whether you are using the namespace mapping (default) or the flattened mapping (deprecated). Previously, applications using the namespace mapping needed to load `Ice_ns.php.`

### Ice Unset

The unset value for optional parameters with the namespace mapping is `\Ice\None` rather than `\Ice\Unset`; the latter cannot be used as `unset` is a PHP keyword. `Ice_Unset` is still available with the flattened mapping.

Back to Top ^

## Python Changes

See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

### AMD

The mapping for asynchronous method dispatch (AMD) has changed significantly. Asynchronous dispatch methods in existing applications will need to be modified as follows:

- Remove the `_async` suffix from the method name
- Remove the callback parameter
- Change how the method reports results and exceptions
- Return an `Ice.Future` object

A dispatch method now has the option of using synchronous semantics or asynchronous semantics. It can return results directly, in which case Ice marshals the results immediately, or it can return a future that the implementation must complete later.

Calls to `ice_response` on the callback object must be converted to calls to `set_result` on the future object. Similarly, calls to `ice_exception` on the callback object must be converted to calls to `set_exception` on the future object.

Consider this operation:

**Slice**

```
string getResults(int id, out bool validated);
```

Suppose we have this existing implementation:

**Python**

```
def getResults_async(self, cb, id, current=None):
    cb.ice_response("answer", True) # Typically done later, e.g., in a separate thread
```

Using Ice 3.7, we need to convert this implementation as follows:

**Python**

```
def getResults(self, id, current=None): # Changed method name, removed callback parameter
    f = Ice.Future()
    f.set_result(("answer", True))      # Typically done later, e.g., in a separate thread
    return f                            # Return a future
```

Pay special attention to the value passed to `set_result`: this method accepts only a single value. If the operation returns multiple values, they must be supplied in a tuple.

Back to Top ^

# Ruby Changes

See also the subsections Changed APIs and Removed APIs for information about APIs defined in Slice and therefore common to all language mappings.

Back to Top ^

# Freeze Persistence Service

The Freeze persistence service, which allows you to store objects defined in Slice in a Berkeley DB database, is no longer bundled with Ice. It is now a separate add-on.

# Migrating the IceGrid and IceStorm Databases from Freeze to LMDB

As of Ice 3.7, IceGrid and IceStorm rely on LMDB for persistent storage. In prior releases, IceGrid and IceStorm were using the Freeze service for persistent storage; Freeze itself stores its data in Oracle Berkeley DB.

Berkeley DB and LMDB are quite similar: they are both embedded database libraries that require little or no administration and configuration. They both maintain persistent key-value maps, where keys and values are sequences of bytes. While Berkeley DB creates many files in its DB environment (one file for each persistent map, log files and more), LMDB creates just two files in its own database enviromnent: a data file (data.mdb) that contains all the persistent maps, and a lock file (lock.mdb). There are no log files with LMDB, which further simplifies administration compared to Berkeley DB.

This section describes how to migrate an IceGrid registry or an IceStorm instance using Ice 3.5 or 3.6 (with a Freeze database) to an IceGrid registry or IceStorm instance using Ice 3.7 (with a LMDB database).

## IceGrid Migration

Prerequisite: you need the IceGrid database export tool version 3.5 (`icegriddb35`) or version 3.6 (`icegriddb`). This utility is included in the Ice 3.6 distribution starting with Ice 3.6.2, but was not included in any Ice 3.5 distribution. If you are migrating from Ice 3.5, you need to build this export tool from sources: icegriddb35.

To start this migration, first stop the IceGrid registry you wish to upgrade, then export the Freeze/Berkeley DB database environment of your IceGrid registry:

**With icegriddb 3.5 or 3.6**

```
icegriddb --export icegriddb.ixp --dbhome /var/icegrid/db
```

The `icegriddb` export tool's version must match the existing IceGrid registry's version; for example, use `icegriddb35` with an IceGrid registry 3.5.x.

The resulting file (`icegriddb.ixp` in our example) is a binary file with the full content of the IceGrid registry database. The `icegriddb` utility can import this file into a Freeze/Berkeley DB or LMDB database.

Next, create a directory for your new IceGrid registry LMDB database files:

```
mkdir /var/icegrid/lmdb
```

Next, import `icegriddb.ixp` into this new LMDB database environment directory:

**With icegriddb 3.7 or greater**

```
icegriddb --import icegriddb.ixp --dbpath /var/icegrid/lmdb
```

Finally, edit your IceGrid registry configuration to replace the `IceGrid.Registry.Data` property with the new `IceGrid.Registry.LMDB.Path` property:

```
IceGrid.Registry.LMDB.Path=/var/icegrid/lmdb
```

While the instructions above are sufficient for most deployments, you may want to review IceGrid Persistent Data and IceGrid Database Utility for detailed information about the tool and LMDB configuration options.

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 3.7 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 <= 3.5 won't interoperate with the IceGrid 3.7 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 3.7 using the instructions above.

Back to Top ^

## IceStorm Migration

Prerequisite: you need the IceStorm database export tool version 3.5 (`icestormdb35`) or version 3.6 (`icestormdb`). This utility is included in the Ice 3.6 distribution starting with Ice 3.6.2, but was not included in any Ice 3.5 distribution. If you are migrating from Ice 3.5, you need to build this export tool from sources: icestormdb35.

To start this migration, first stop the IceStorm server you wish to upgrade.

Then export the Freeze/Berkeley DB database environment of your IceStorm server:

**With icestormdb 3.5 or 3.6**

```
icestormdb --export icestormdb.ixp --dbhome /var/icestorm/db
```

The `icestormdb` export tool's version must match the existing IceStorm version; for example, use `icestormdb35` with an IceStorm 3.5.x.

The resulting file (`icestormdb.ixp` in our example) is a binary file with the full content of the IceStorm database.

If you deployed IceStorm with IceGrid, the IceStorm database environment is typically specified through a Freeze `dbenv` descriptor, and the corresponding Berkeley DB home directory is in a subdirectory of your IceGrid node data directory.

The `icestormdb` utility can import this file into a Freeze/Berkeley DB or LMDB database.

Next, create a directory for your new IceStorm LMDB database files:

```
mkdir /var/icestorm/lmdb
```

Now import `icestormdb.ixp` into this new LMDB database environment directory:

**With icestormdb 3.7 or greater**

```
icestormdb --import icestormdb.ixp --dbpath /var/icestorm/lmdb
```

Finally, edit your IceStorm configuration file and replace the `Freeze.DbEnv.<Service Name>.DBHome` property with the new property `<Service Name>.LMDB.Path`.

When IceStorm is deployed through IceGrid, a typical and recommended directory for this LMDB database is `${service.data}`.

While the instructions above are sufficient for most deployments, you may want to review IceStorm Persistent Data and IceStorm Database Utility for detailed information about the tool and LMDB configuration options.

Back to Top ^

# Changed APIs

This section describes APIs common to multiple language mappings (often specified using Slice) that have changed, potentially in ways that are incompatible with previous releases.

## Forward Declared Slice Interfaces and Classes

A forward declared interface or class must be fully defined in the same Slice translation unit if any Slice definition in this translation unit uses a proxy of this interface (or class), or uses this forward declarated class in a non-local context (typically as a parameter in a non local operation).

## Connection and Endpoint Information

The local classes `Ice::EndpointInfo` and `Ice::ConnectionInfo`, and all derived classes (`Ice::IPEndpointInfo`, `IceSSL::EndpointInfo`, etc.) were refactored. These classes now support an `underlying` data member that provides information on the underlying transport. For example, the `ssl` transport is based on the `tcp` transport so the `underlying` data member of an `ssl` endpoint or connection will contain an instance of `Ice::TCPEndpointInfo` or `Ice::TCPConnectionInfo`. See Using Connections for additional information.

## Connection Changes

The API for Ice::Connection has changed in several ways:

- there are now separate callbacks for the close and heartbeat callbacks
- `Connection::close` now accepts an enum parameter instead of a bool

## Flushing Batch Requests

These operations now take an additional parameter to control compression.

## Classes no longer derive from Object

(Affects all language mappings except: C++98, Java Compat, Objective-C)

In Ice 3.6 and prior releases, a Slice class derives implicitly from `Object`, just like Slice interfaces. In Ice 3.7, Slice classes derive implicitly from `Value` (a new keyword). Slice interfaces still implicitly inherit from `Object`.

When mapped to C#, C++ with the C++11 mapping, Java, JavaScript, Python and more, the corresponding mapped native class no longer derives from `Ice::Object`. It derives instead from `Ice::Value`. Let's take an example:

**Slice**

```
module M
{
    class A
    {
        string x;
    };
};
```

This Slice class `A` is mapped to:

**C#**

```
// 3.6
public partial class A : Ice.Object
{
    ...
}

// 3.7
public partial class A : Ice.Value
{
    ...
}
```

**JavaScript**

```
// 3.6
M.A = class extends Ice.Object
{
    ...
};

// 3.7
M.A = class extends Ice.Value
{
    ...
};
```

**Python**

```
# 3.6
class A(Ice.Object):
    ...

# 3.7
class A(Ice.Value):
    ...
```

In the language mappings unaffected by this change - C++98, Java Compat and Objective-C - `Value` and `Object` are mapped to the same native class.

Moreover, an operation on a Slice class is no longer mapped to an abstract method on the corresponding native class: the Slice compiler generates instead a separate skeleton class (typically with a `Disp` suffix) with the mapped method.

In a similar fashion, the mapped class for a Slice class that implements an interface no longer implements anything related to this interface. The Slice compiler generates instead a separate, independent skeleton class that implements the mapped interface.

## Interface Operation Parameters

(Affects all language mappings except: C++98, Java Compat, Objective-C)

In Ice 3.6 and prior releases, you could use an interface as the type for an operation parameter, for example:

**Slice**

```
module M
{
    interface Marker
    {
        string print();
    };

    interface Receiver
    {
        Marker op(Marker x); // Marker not Marker*, i.e. pass-by-value
    };

    class A implements Marker
    {
        string msg;
    };
};
```

This way, only instances of Slice classes that implement this interface would be accepted as a parameter to this operation.

With Ice 3.7, the Slice definitions above remain valid, but the parameters are now mapped like `Values`. For example, the `Receiver` interface is mapped to the following proxy and skeleton classes in C#:

**C#**

```
// Proxy
public interface ReceiverPrx : Ice.ObjectPrx
{
    Ice.Value op(Ice.Value x, Ice.OptionalContext context = new Ice.OptionalContext());
    ...
}

// Skeleton
public abstract class ReceiverDisp_ : Ice.ObjectImpl, Receiver
{
    public abstract Ice.Value op(Ice.Value x, Ice.Current current = null);
    ...
}
```

In the unusual situation where you need to send an interface "instance" by value, where the value carries only the interface's type id, each language mapping provides a new helper class for this purpose named `InterfaceByValue`.

Back to Top ^

# Removed APIs

The following APIs were removed in this release:

- Interface `Ice::ConnectionCallback` (replaced by `Ice::HeartbeatCallback` and `Ice::CloseCallback`)
- Operation `Ice::Connection::setCallback` (replaced by the `setCloseCallback` and `setHeartbeatCallback` operations)
- Exception `Ice::NoObjectFactoryException` (replaced by `Ice::NoValueFactoryException`)
- Exception `Ice::ForcedCloseConnectionException` (replaced by `Ice::ConnectionManuallyClosedException`)
- Class `Ice::UnknownSlicedObject` (replaced by `Ice::UnknownSlicedValue`)

Back to Top ^

# Deprecated APIs

This section describes the APIs that are deprecated in this Ice release, and will be removed in a future release. If your application uses one or more of the APIs listed below, we recommend updating it as soon as possible.

## Operations on Classes

Operations on Slice classes are now deprecated: when feasible, you should convert these classes to interfaces (with only operations and no data members) or to classes without operations. If you need to keep classes with operations for interoperability with older applications, the global metadata directive `suppress-warning:deprecated` allows you to compile your Slice files without warnings.

Likewise, having a Slice class implement one or more interfaces is now deprecated.

Local classes with operations are not deprecated. Such local classes are used by some Ice APIs, such as `Ice::EndpointInfo`.

## Object Factories

Object factories are now referred as value factories following the deprecation of classes with operations. As a result, the following `Ice::Communicator` operations have been deprecated:

- `Communicator::addObjectFactory`
- `Communicator::findObjectFactory`

You should now use the `ValueFactoryManager` interface returned by `Communicator::getValueFactoryManager` to manage value factories.

## Exception `ice_name` Method

The `ice_name` method for Ice exceptions has been deprecated in the various language mappings. It has been replaced by a new `ice_id` method.

## Thread Hook in Python and C#

The `threadHook` member of `InitializationData` is deprecated. The new members `threadStart` and `threadStop` can be set to callable objects (Python) or `System.Action` delegates (C#).

## Batch Request Interceptor in Python

The `Ice.BatchRequestInterceptor` class is deprecated. The `batchRequestInterceptor` member of `InitializationData` can be set to a callable object.

## IcePatch2

IcePatch2 and IceGrid's distribution mechanism (based on IcePatch2) are now deprecated.

Back to Top ^