New Features in Ice 3.4

This page describes significant changes and improvements in Ice 3.4 that may affect the operation of your applications or have an impact on your source code. For a detailed list of the changes in this release, please refer to the `CHANGES` file included in your Ice distribution.

On this page:

- Changes and fixes in Ice 3.4.2
  - Supported platforms
  - Ice Visual Studio Add-in
  - .NET Compact Framework support
  - Android support
- Changes and fixes in Ice 3.4.1
  - Underscores allowed in Slice
    - Interoperability
    - Name collisions
    - Freeze
  - Freeze locking
  - Miscellaneous changes
- Features added with Ice 3.4.0
  - New API for Asynchronous Method Invocation (AMI)
  - Better scalability
  - New Dispatcher facility
  - Glacier2 utility classes
  - Default servants
  - Alternate storage for IceGrid and IceStorm
  - Connection and endpoint information
  - New Slice compiler and API for PHP
  - Slice comments
  - New Slice syntax for default values
  - Properties in the Windows registry
  - New sample programs

Changes and fixes in Ice 3.4.2

Supported platforms

We are adding Red Hat Enterprise Linux 6 (i386 and x86_64) and Amazon Linux 2011.2 (i386 and x86_64) to our list of supported platforms for Ice 3.4.2. Binary RPMs for RHEL 6 are available for download; these RPMs can also be used on Amazon Linux.

Ice Visual Studio Add-in

We have made many more improvements to our Visual Studio Add-in. This release adds support for .NET Compact Framework projects and includes several bug fixes and minor enhancements to improve the user experience. Refer to the `CHANGES` file included in your Ice distribution for more details.

.NET Compact Framework support

Ice for .NET now includes support for the .NET Compact Framework (.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 binary distribution 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`.

Due to API limitations, 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`
- Automatic discovery of dependent assemblies containing Slice-generated classes and exceptions

Refer to the manual for more details.

Android support

Support for using Ice in Android applications was previously provided as a ZeroC Labs project. This project consisted of source code patches to the Ice for Java run time that corrected compatibility issues, along with several sample applications that demonstrated the use of Ice in Android.

The source code patches have been incorporated into the Ice mainline for the 3.4.2 release, meaning you can now use the same `Ice.jar` file to build both Java and Android applications. The Android sample projects have also been added to the Ice distribution.

Ice requires Android 2.1 or later, and Android 2.2 or later is required to use IceSSL.

Changes and fixes in Ice 3.4.1

Underscores allowed in Slice

Prior versions of Ice did not permit underscores to be used in Slice identifiers. We have eliminated that restriction in Ice 3.4.1 with a new translator option (`--underscore`).

Please note that there are several important issues to consider if you plan to incorporate underscores into your Slice definitions:

Interoperability

Renaming an existing Slice definition always raises the possibility of interoperability problems with existing applications. Changing the name of any Slice definition whose type ID is sent "over the wire" can easily break interoperability unless all applications are rebuilt and redeployed. Adding underscores to your Slice definitions presents an additional difficulty because the Slice compilers for older versions of Ice will not even be able to compile your new definitions.

Name collisions

With some effort, it is possible to write legal Slice definitions using underscores that generate name collisions in a language mapping. For example:

Slice

```
module A
{
    interface B_C { };
};

module A_B
{
    interface C { };
};
```

The Slice compiler for PHP (`slice2php`) uses underscores to separate name scopes in the flattened mapping, therefore both of these interfaces generate the same PHP type named `A_B_C`.

Here is another example:

Slice

```
module A
{
    interface B
    {
        void op();
        void begin_op();
    };

    struct Callback_B_op
    {
        string s;
    };
};
```

These Slice definitions cause collisions with generated code that supports asynchronous invocations.

Although these are contrived examples written intentionally to cause errors, they highlight the importance of selecting your Slice identifiers with careful consideration of your target language mappings.

Freeze

As discussed above, renaming Slice types poses a range of compatibility issues. If you use Freeze to store instances of Slice types persistently, be aware that renaming Slice types usually requires that you also migrate your Freeze databases because Slice type names are embedded in your records (if you store instances of Slice classes) and also appear in the Freeze catalog.

Freeze locking

Ice 3.4.0 added a locking mechanism to Freeze to prevent multiple processes from opening the same database environment simultaneously, which can lead to data corruption. Freeze uses a lock file named `__Freeze/lock`, which can be found in the database environment directory.

In Ice 3.4.1 we added the property `Freeze.DbEnv.`*`env-name`*`.LockFile`. This property determines whether Freeze attempts to create the lock file for the named database environment. The default value of this property is 1, meaning the lock file is created. Applications should not normally need to disable the lock file, but it is useful for utility programs such as the FreezeScript tools dumpdb and transformdb. By disabling the lock file, these tools are able to inspect a database environment that is currently open in another process, regardless of whether that process created a lock file.

If you intend to use a FreezeScript tool on a database environment that is currently open, please be aware that the property `Freeze.DbEnv.`*`env-name`*`.DbPrivate=0` must be defined for both the FreezeScript tool as well as the other process that has opened the environment, otherwise the database can be corrupted.

Miscellaneous changes

- Ice 3.4.1 requires Berkeley DB 4.8.30. This version of Berkeley DB includes a fix for a minor memory leak that was present in earlier versions.
- The shrinking behavior of Ice thread pools changed in Ice 3.4.0 but was not documented. Users of both 3.4.0 and 3.4.1 should review Thread pool changes in Ice 3.4 for more information.

Features added with Ice 3.4.0

New API for Asynchronous Method Invocation (AMI)

This release features a completely new AMI facility for C++, C#, Java, and Python that allows you to structure your code with much greater flexibility. To get a better understanding of the motivations for this enhancement and how it can improve your own applications, we encourage you to read our white papers on AMI.

Better scalability

The Ice run time underwent significant retooling to make use of Windows completion ports and overlapped I/O for its networking operations. As a result, server applications that handle many connections should see a significant improvement in scalability.

Additional enhancements were made to improve scalability on all platforms. For example, Ice now establishes an outgoing connection and accepts an incoming connection in constant time, regardless of the number of connections that have already been established. Furthermore, the Ice thread pool now supports receiving and sending data using multiple threads, which improves CPU usage on machines with multiple cores.

Our performance white paper has been updated for Ice 3.4 and shows that the scalability of Ice for C++ on Windows is now on par with the scalability on Linux.

New `Dispatcher` facility

In previous releases, the developer of a graphical Ice application would need to take precautions to make sure that updates to the user interface were performed in the proper thread. For example, graphical applications typically use AMI because it does not block the calling thread, but AMI callbacks are invoked from an Ice run time thread. Since the callback cannot update the user interface directly from such a thread, it is forced to schedule an update instead. Consequently, the application code grew more complex and was prone to error if the developer neglected the threading rules.

Ice 3.4 introduces the `Dispatcher` facility that lets you control the thread in which servant methods and AMI callbacks are invoked. It is especially useful for a graphical application, in which you can easily install a custom dispatcher to guarantee that all of your servant and callback invocations are made in a thread that can safely update the user interface.

This technique is demonstrated in a sample application for each of the language mappings: refer to the `MFC` demo in C++, the `swing` demo in Java, and the `wpf` demo in C#.

Glacier2 utility classes

Some effort is required to write an application that correctly manages a Glacier2 session. To simplify this task, we have added utility classes in C++, C#, Java, and Python that manage the session for you.

For all of these language mappings, Ice includes a new subclass of `Ice::Application` named `Glacier2::Application` that is intended to be used by command-line applications that require a Glacier2 session. The class takes care of establishing the session, keeping it alive, and recovering from session failures.

In Java and C#, Ice also includes the `Glacier2.SessionHelper` class for use in graphical applications. This class performs many of the same tasks as `Glacier2.Application`.

Default servants

The traditional way of implementing a default servant was to install a servant locator that returned the same servant for every request. However, since default servants are one of the most common use cases for servant locators, we have made them easier to use by adding new operations to the `ObjectAdapter` interface. If you make use of default servants, you can simplify your code by migrating your application to this new API.

Alternate storage for IceGrid and IceStorm

IceStorm and the IceGrid registry now have the ability to use a SQL database instead of Freeze for their persistent storage requirements. This release supports SQLite on all platforms and PostgreSQL on Unix platforms. If you would like to see support added for other SQL servers, please contact info@zeroc.com.

Connection and endpoint information

Ice developers frequently want to obtain information about connections and endpoints, as evidenced by the numerous inquiries on the user forum about this issue. For example, the developer may want to be able to discover the IP address of a remote client. This information was available in prior releases, but not in a form that was easy to manipulate.

Now it is possible to obtain more details about connections and endpoints with the addition of the `ConnectionInfo` and `EndpointInfo` class hierarchies. Using these classes, you can discover addresses, ports, and other attributes of a connection or endpoint.

Note that it is inadvisable to use such addressing information for authentication purposes, as IP addresses can easily be forged.

New Slice compiler and API for PHP

Significant changes have been made to the PHP mapping and API. For example, Ice for PHP now uses static translation via the new `slice2php` compiler, and the language mapping is now more consistent with that of Python and Ruby. For more information on migrating your PHP application, please refer to PHP changes in Ice 3.4.

Slice comments

This release offers several improvements that will be appreciated by developers, especially those who use integrated development environments (IDEs):

- The Slice compilers for Java, C#, and Python now preserve Slice comments in the generated code.
- Doc comments have been added to the native Ice APIs.
- The JAR files for Ice and Freeze include source code to allow IDEs such as Eclipse to browse the Ice source code and to display javadoc comments.
- For C# users that compile generated Slice code into an assembly, it is now useful to instruct Visual C# to emit documentation comments into an XML file for the assembly containing your compiled Slice definitions. This enables the IDE to display tooltips for your Slice APIs. Ice generates these XML files for its own assemblies so that you can view tooltips for the Ice APIs as well.

New Slice syntax for default values

It is now possible to specify in Slice the default values for data members of classes, structures, and exceptions. The semantics are the same as for Slice constants in that you can only specify default values for a data member whose type is a primitive or enumeration. For example:

Slice

```
enum Color { red, green, blue };

struct Point
{
    int x = -1;
    int y = -1;
    Color c = blue;
};
```

Properties in the Windows registry

Ice configuration properties can now be loaded from the Windows registry by specifying a registry key as the value of the `Ice.Config` property. Ice programs that run as Windows services are likely to make use of this feature because it avoids the need to hard-code properties in the application, eliminates the dependency on a configuration file, and allows the program's configuration settings to be edited using familiar registry tools.

New sample programs

This release adds the following sample programs:

- `map_filesystem` (C++, Java)
  Shows how to implement the file system application using a Freeze map.
- `interleaved` (C++)
  Uses interleaved asynchronous invocations to achieve maximum throughput.
- `plugin` (C++, Java, C#)
  Demonstrates how to write an Ice plug-in.
- `chat` (Java, C#)
  A graphical chat client that uses Glacier2 to communicate with a C++ server.
- `swing` (Java) and `wpf` (C#)
  Graphical versions of the `hello` client.
- `hello` (PHP)
  A Glacier2 client that demonstrates the use of registered communicators.

See Also