Release Notes

The release notes provide information about an Ice-E release, including descriptions of significant new features and changes, instructions for upgrading from an earlier release, and important platform-specific details.

On this page:

Back to Top ^

New Features in Ice-E 1.3

This section describes the significant new features in Ice-E 1.3.

Thread pool concurrency model

Ice-E now supports the same thread pool concurrency model as Ice. The communicator creates two thread pools:

- The client thread pool is created when the communicator is initialized and services outgoing connections.
- The server thread pool is created upon object adapter creation and services incoming connections.

Ice-E does not support per-object adapter thread pools. Thread pools are configured with the same properties as Ice thread pools. For more information on thread pools, please refer to The Ice Threading Model.

Back to Top ^

Non-blocking AMI

Ice-E now supports asynchronous message invocation with the same semantics as Ice 3.3. In particular, Ice-E guarantees that the calling thread will never block when issuing an asynchronous invocation. The non-blocking nature of AMI requests is especially useful for a graphical application.

AMI in Ice-E supports both oneway and twoway requests and it also supports the same API as Ice to enable an application to implement its own flow control logic.

Please refer to Asynchronous Programming for more information on the AMI semantics.

Back to Top ^

Real-time support

Ice-E supports new real-time features.

A new `start` method in `IceUtil::Thread` allows you to start a thread with a given priority. On POSIX systems, real-time threads use the `SCHED_RR` scheduling policy.

Ice-E mutexes can also be configured to use the priority inheritance protocol on POSIX systems. This protocol is useful to avoid thread priority inversion. The mutex protocol used by default is specified by the `DEFAULT_MUTEX_PROTOCOL` macro defined in the `cppe/config/Make.rules` file.

Please refer to the `README` file for more information on Ice-E real-time features.

Back to Top ^

Default servants

This Ice-E release introduces a new default servant API. When combined with dispatch interceptors, default servants can be used as a replacement for servant locators, which are not supported in Ice-E.

Default servants dispatch requests for Ice objects that are not registered with the object adapter's Active Servant Map (ASM). A default servant is registered with an object adapter for a given category.

Back to Top ^

Objects by value

It's now possible to pass Slice classes by value. Ice-E supports transmitting abstract and concrete Slice classes (with and without operations, respectively). As in Ice, you're responsible for registering an object factory with the communicator to be able to receive a Slice class that has operations.

Ice-E does not support garbage collection of Slice class instances. If your Slice classes contain cyclic dependencies, you will get memory leaks if you fail to break the dependencies between objects.

Back to Top ^

Dynamic network interfaces

Ice-E server endpoints that are configured to listen on all local interfaces now listen on `INADDR_ANY/0.0.0.0` rather than listening separately on each of the interfaces that were found at the time the server started. This means that if a new interface becomes available while the server is running, Ice-E will be able to receive and process requests using that interface.

Note that, if an interface change occurs, an object adapter does not automatically refresh the list of endpoints that it embeds in newly-created proxies. Rather, an application must instruct the object adapter to refresh its endpoint list using the new operation `refreshPublishedEndpoints`.

Back to Top ^

Dispatch interceptors

A dispatch interceptor is a server-side mechanism that allows you to intercept incoming client requests before they are given to a servant. The interceptor can examine the incoming request; in particular, it can examine the `Current` information for the request (which provides access to the operation name, object identity, and so on).

A dispatch interceptor can dispatch a request to a servant and check whether the dispatch was successful; if not, the interceptor can choose to retry the dispatch. This functionality is useful to automatically retry requests that have failed due to a recoverable error condition.

Back to Top ^

Platform notes for Ice-E 1.3

This section lists the platforms supported by Ice-E, including those platforms that were added or removed in the 1.3 release.

Supported platforms

The following operating systems are supported for Ice-E:

- Red Hat Enterprise Linux 5 update 2 (RHEL 5.2) i386 and x86_64 *
- Windows XP and Vista x86
- Windows Vista x64 *
- Windows Mobile 6 Professional (ARMV4I)
- Gumstix Linux buildroot revision 1364

  * *New for this release.*

The following C++ compilers are supported:

- GCC 4.1.2 on RHEL 5.2
- GCC 4.1.1 ARM cross-compiler for Gumstix
- Microsoft Visual C++ 2005 SP1 Professional
- Microsoft Visual C++ 2008 Professional *

  * *New for this release.*

Back to Top ^

Desupported platforms

Ice-E no longer supports the following operating systems:

- Windows Mobile 2003
- Windows Mobile 5.0
- Red Hat Enterprise Linux 4 update 4 (i386 and x86_64)
- SuSE Linux Enterprise Server 10 (i586 and x86_64)

Back to Top ^

Fixes and improvements in Ice-E 1.3

Below is an abbreviated list of changes included in this release that were not already discussed above. For a complete list of changes, please refer to the `CHANGES` file included in your Ice-E distribution.

- A new object adapter property, `<adapter>.ProxyOptions`, lets you customize the proxies that the adapter creates.

- The new property `Ice.TCP.Backlog` allows you to specify the size of incoming connection backlog for TCP/IP sockets. This setting is also used for SSL. In C++ the default value is `SOMAXCONN` or 511 if that macro is not defined.

- Batch requests can be flushed for individual proxies using two new proxy methods:

  C++

  ```
  void ice_flushBatchRequests()
  bool ice_flushBatchRequests_async(callback)
  ```

  As its name implies, the second method allows you to flush batch requests with non-blocking AMI semantics.

- If a proxy contains a host that is multihomed, the client will now try all of the IP addresses reported by DNS. Previously, only the first address in the list was used and others were ignored.

- Most proxy factory methods now return a proxy of the same type as the original and no longer require the use of a checked or unchecked cast. For example, in C++ you can write

  C++

  ```
  HelloPrx hello = ...;
  hello = hello->ice_oneway();
  ```

  Previously you would have needed a cast, such as

  ```
  hello = HelloPrx::uncheckedCast(hello->ice_oneway());
  ```

- The rules for legal property names and values have been relaxed. Refer to Configuration File Syntax for a complete description of the syntax. The limit on the line length of a property in a configuration file has also been removed.

- Added support for opaque endpoints in stringified proxies, which allows you to preserve the endpoint of an unknown transport (such as an SSL endpoint when IceSSL is not installed).

Back to Top ^

Desupported components in Ice-E 1.3

This section describes Ice-E features and components that are no longer supported.

Thread-per-connection concurrency model

The new thread pool concurrency model, together with the addition of the non-blocking AMI feature, have made the thread-per-connection concurrency model unecessary and it has therefore been removed from this release. See below for additional information.

Back to Top ^

Blocking concurrency model

The blocking concurrency model has been removed from this release. The blocking concurrency model was only useful for performance purposes. With the addition of AMI and the thread pool concurrency model, supporting this concurrency model would have been too complex.

Back to Top ^

Java language mapping

Ice-E no longer support the Java mapping. Only the C++ mapping is supported.

Back to Top ^

Upgrading your application to Ice-E 1.3

Removed APIs

This section describes features and APIs that are no longer available in this release.

Thread-per-connection concurrency model

One useful feature of this concurrency model was serializing the requests received over a connection. This was helpful when the application needed to ensure that requests are dispatched in the order they are received, or when the application did not want to implement the synchronization that might be required by using the thread pool concurrency model.

The ability to serialize requests is now provided by the thread pool and enabled via the configuration property `<threadpool>.Serialize`.

Back to Top ^

Blocking concurrency model

This concurrency model is no longer supported. As a result, the `Ice.Blocking` configuration property should be removed from your configuration files.

Back to Top ^