Ice Miscellaneous Properties

On this page:

Ice.ACM.Client

Synopsis

`Ice.ACM.Client=`*`num`*

Description

If *`num`* is set to a value larger than zero, client-side Active Connection Management (ACM) is enabled. This means that connections are automatically closed by the client after they have been idle for *`num`* seconds. This is transparent to applications because connections closed by ACM are automatically reestablished if they are needed again. The default value is `60`, meaning that idle connections are automatically closed after one minute.

Applications may need to disable client-side ACM in certain situations, such as when using bidirectional connections. You can disable ACM by setting this property to zero.

Ice.ACM.Server

Synopsis

`Ice.ACM.Server=`*`num`*

Description

This property is the server-side equivalent of `#Ice.ACM.Client`. If *`num`* is set to a value larger than zero, server-side Active Connection Management (ACM) is enabled, in which the server automatically closes an incoming connection after it has been idle for *`num`* seconds. The default value is `0`, meaning that server-side ACM is disabled by default.

Server-side ACM can cause incoming oneway requests to be silently discarded.

Ice.BackgroundLocatorCacheUpdates

Synopsis

`Ice.BackgroundLocatorCacheUpdates=`*`num`*

Description

If *`num`* is set to zero (the default), an invocation on an indirect proxy whose endpoints are older than the configured locator cache timeout triggers a locator cache update; the run time delays the invocation until the new endpoints are returned by the locator.

If *`num`* is set to a value larger than zero, an invocation on an indirect proxy with expired endpoints still triggers a locator cache update, but the update is performed in the background, and the run time uses the expired endpoints for the invocation. This avoids delaying the first invocation that follows expiry of a cache entry.

Ice.BatchAutoFlush

Synopsis

`Ice.BatchAutoFlush=`*`num`*

Description

This property controls how the Ice run time deals with flushing of batch messages. If *`num`* is set to a non-zero value (the default), the run time automatically forces a flush of the current batch when a new message is added to a batch and that message would cause the batch to exceed `#Ice.MessageSizeMax`. If *`num`* is set to zero, batches must be flushed explicitly by the application; in this case, if the application adds more messages to a batch than permitted by `Ice.MessageSizeMax`, the application receives a `MemoryLimitException` when it flushes the batch.

Ice.CacheMessageBuffers

Synopsis

`Ice.CacheMessageBuffers=`*`num`* (Java, .NET)

Description

If *`num`* is a value greater than zero, the Ice run time caches message buffers for future reuse. This can improve performance and reduce the amount of garbage produced by Ice internals that the garbage collector would eventually spend time to reclaim. However, for applications that exchange very large messages, this cache may consume excessive amounts of memory and therefore should be disabled by setting this property to zero.

This property affects the caching of message buffers for synchronous invocations. The Ice run time never caches message buffers for asynchronous invocations.

Platform Notes

- Java
  Ice allocates non-direct message buffers when this property is set to 1 and direct message buffers when set to 2. Use of direct message buffers minimizes copying and typically results in improved throughput. If not defined, the default value is 2.
- .NET
  If not defined, the default value is 1.

Ice.ChangeUser

Synopsis

`Ice.ChangeUser=`*`user`* (C++ & Unix only)

Description

If set, Ice changes the user and group id to the respective ids of *`user`* in `/etc/passwd`. This only works if the Ice application is executed by the super-user.

Ice.ClientAccessPolicyProtocol

Synopsis

`Ice.ClientAccessPolicyProtocol=`*`type`* (Silverlight only)

Description

If set to `Http`, Ice for Silverlight configures an outgoing socket to use the HTTP access policy protocol rather than the default TCP protocol.

Ice.Compression.Level

Synopsis

`Ice.Compression.Level=`*`num`*

Description

Specifies the bzip2 compression level to use when compressing protocol messages. Legal values for *`num`* are `1` to `9`, where `1` represents the fastest compression and `9` represents the best compression. Note that higher levels cause the bzip2 algorithm to devote more resources to the compression effort, and may not result in a significant improvement over lower levels. If not specified, the default value is `1`.

Ice.ConsoleListener

Synopsis

`Ice.ConsoleListener=`*`num`* (.NET)

Description

If *`num`* is non-zero, the Ice run time installs a `ConsoleTraceListener` that writes its messages to `stderr`. If *`num`* is zero, logging is disabled. Note that the setting of `#Ice.LogFile` overrides this property: if `Ice.LogFile` is set, messages are written to the log file regardless of the setting of `Ice.ConsoleListener`.

Ice.EventLog.Source

Synopsis

`Ice.EventLog.Source=`*`name`* (C++ & Windows only)

Description

Specifies the name of an event log source to be used by a Windows service that subclasses `Ice::Service`. The value of *`name`* represents a subkey of the `Eventlog` registry key. An application (or administrator) typically prepares the registry key when the service is installed. If no matching registry key is found, Windows logs events in the `Application` log. Any backslashes in *`name`* are silently converted to forward slashes. If not defined, `Ice::Service` uses the service name as specified by the `--service` option.

Ice.FactoryAssemblies

Synopsis

`Ice.FactoryAssemblies=`*`assembly`* `[`*`assembly ...`*`]` (Silverlight, .NET CF)

Description

The Silverlight and Compact Framework versions of Ice for .NET are unable to automatically discover all dependent assemblies during program startup. To assist the Ice run time in locating user exceptions and concrete Slice classes, you must explicitly list the assemblies that contain Slice-generated code. The value of this property is a whitespace-separated list of assembly names, which may be simple names (such as `client`) or fully-qualified names (such as `client,Version=1.0.0.0,Culture=neutral,PublicKeyToken=...`). Do not use spaces in a fully-qualified assembly name.

When searching for a class, Ice first checks in the assemblies specified by this property. If the type is not found, Ice automatically looks in the standard Ice assemblies (`Ice`, `Glacier2`, `IceBox`, `IceGrid`, `IcePatch2`, and `IceStorm`). This means it is not necessary for you to explicitly include the standard Ice assemblies in this property.

Note that the program itself is also considered an assembly. If you compiled the main program directly with Slice-generated code, your `Ice.FactoryAssemblies` property must include the program itself if the generated code includes user exceptions or concrete classes. For simple build scenarios in which all generated code is compiled directly into the executable, the following configuration setting is sufficient:

```
Ice.FactoryAssemblies=client
```

This example assumes the executable is named `client.exe`. On the other hand, if Slice-generated code is also compiled into a dependent assembly, your configuration might look like this instead:

```
Ice.FactoryAssemblies=client MyOtherAssembly
```

Failing to define `Ice.FactoryAssemblies` can cause the Ice run time in the receiver to raise `NoObjectFactoryException`, `UnmarshalOutOfBoundsException` or `UnknownUserException`. If you are experiencing either of these exceptions, verify that your assemblies are configured correctly.

Ice.GC.Interval

Synopsis

`Ice.GC.Interval=`*`num`* (C++)

Description

Ice for C++ includes a garbage collector for Slice objects. This property determines the frequency with which the garbage collector runs. If the interval is set to zero (the default), no collector thread is created. Otherwise, the collector thread runs every *`num`* seconds. You can trace the garbage collector's activities by setting the `Ice.Trace.GC` property.

Ice.ImplicitContext

Synopsis

`Ice.ImplicitContext=`*`type`*

Description

Specifies whether a communicator has an implicit request context and, if so, at what scope the context applies. Legal values for this property are `None` (equivalent to the empty string), `PerThread`, and `Shared`. If not specified, the default value is `None`.

Ice.LogFile

Synopsis

`Ice.LogFile=`*`file`*

Description

Replaces the communicator's default logger with a simple file-based logger implementation. This property does not affect the per-process logger. The logger creates the specified file if necessary, otherwise it appends to the file. If the logger is unable to open the file, the application receives an `InitializationException` during communicator initialization. If a logger object is supplied in the `InitializationData` argument during communicator initialization, it takes precedence over this property.

The logger does not provide any built-in support for log file maintenance (such as log rotation), but it can coexist with system tools such as `logrotate`.

Ice.MessageSizeMax

Synopsis

`Ice.MessageSizeMax=`*`num`*

Description

This property controls the maximum size (in kilobytes) of an uncompressed protocol message that is accepted or sent by the Ice run time. The size includes the size of the Ice protocol header. The default size is `1024` (`1` Megabyte). Settings with a value less than `1` are ignored.

The only purpose of this property is to prevent a malicious or buggy client or server from triggering a large memory allocation in another server or client. If this is not a concern, you can set `Ice.MessageSizeMax` to a large value; doing so has no effect on memory allocation or performance.

If a client sends a message that exceeds the client's `Ice.MessageSizeMax`, or the server returns a reply that exceeds the client's `Ice.MessageSizeMax`, the client receives a `MemoryLimitException`.

If a client sends a message that exceeds the server's `Ice.MessageSizeMax`, the server immediately closes its connection, so the client receives a `ConnectionLostException` in that case. In addition, the server logs a `MemoryLimitException` if `Ice.Warn.Connections` is set.

If the server returns a reply that exceeds the server's `Ice.MessageSizeMax`, the server logs a `MemoryLimitException` (if `Ice.Warn.Connections` is set) but does not close its connection to the client. The client receives an `UnknownLocalException` in this case.

Ice.MonitorConnections

Synopsis

`Ice.MonitorConnections=`*`num`*

Description

If Active Connection Management (ACM) is enabled, the Ice run time scans for idle connections to be closed once every *`num`* seconds. If you do not set this property or set *`num`* to zero or a negative value, the run time applies a heuristic to determine how often to scan for idle connections: the default scanning interval is the 10% of smallest configured ACM timeout, with a minimum of 5 seconds, and a maximum of 5 minutes.

Ice.Nohup

Synopsis

`Ice.Nohup=`*`num`*

Description

If *`num`* is set to a value larger than zero, the `Application` convenience class (as well as the `Ice::Service` class in C++) ignore `SIGHUP` on Unix and `CTRL_LOGOFF_EVENT` on Windows. As a result, an application that sets `Ice.Nohup` continues to run if the user that started the application logs off. The default value for `Application` is `0`, and the default value for `Ice::Service` is `1`.

Ice.NullHandleAbort

Synopsis

`Ice.NullHandleAbort=`*`num`*

Description

If *`num`* is set to a value larger than zero, invoking an operation using a null smart pointer causes the program to abort immediately instead of raising `IceUtil::NullHandleException`.

Ice.Package.*module*

Synopsis

`Ice.Package.`*`module=package`* (Java)

Description

Ice for Java allows you to customize the packaging of generated code. If you use this feature, the Ice run time requires additional configuration in order to successfully unmarshal exceptions and concrete class types. This property associates a top-level Slice *module* with a Java *package*. If all top-level modules are generated into the same user-defined package, it is easier to use `Ice.Default.Package` instead.

Ice.PrintAdapterReady

Synopsis

`Ice.PrintAdapterReady=`*num*

Description

If *num* is set to a value larger than zero, an object adapter prints "*adapter_name* ready" on standard output after activation is complete. This is useful for scripts that need to wait until an object adapter is ready to be used.

Ice.PrintProcessId

Synopsis

`Ice.PrintProcessId=`*num*

Description

If *num* is set to a value larger than zero, the process ID is printed on standard output upon startup.

Ice.PrintStackTraces

Synopsis

`Ice.PrintStackTraces=`*num* (C++ only)

Description

If *num* is set to a value larger than zero, inserting an exception that derives from `IceUtil::Exception` into a logger helper class (such as `Ice::Warning`) also displays the exception's stack trace. Likewise, the `ice_stackTrace` function on the base exception class, `IceUtil::Exception`, will return the stack trace or an empty string depending on the value *num*. If not set, the default value depends on how the Ice run time is compiled: 0 for an optimized build and 1 for a debug build.

Stack traces are currently not available on Solaris.

On Windows, stack traces are available in debug builds and in release builds built with the environment variable `RELEASEPDBS=yes`. The release DLLs included in the standard Ice binary distribution are built with this setting enabled. The `.PDB` files for both the debug and release distribution builds are available in a separate Ice PDBs Windows installer.

When generating a stack trace, Windows will attempt to locate the `.PDB` files at the location stored in the associated DLLs; in case the `.PDB` file is not found at this location, Windows will attempt to locate this file using the following search path: current working directory, then the directory of the `IceUtil` DLL, and finally the search path specified by the environment variable `_NT_SYMBOL_PATH`. Therefore, if your `.PDB` files are no longer where you built them, you will need to copy them next to the `IceUtil` DLL or include their new directory in the `_NT_SYMBOL_PATH` environment variable.

This property is only supported in C++ and the scripting languages that use the Ice for C++ run time (Python, Ruby, PHP). Be aware that enabling this property in Python, Ruby or PHP will only display C/C++ stack traces.

Ice.ProgramName

Synopsis

`Ice.ProgramName=`*name*

Description

name is the program name, which is set automatically from `argv[0]` (C++) and from `AppDomain.CurrentDomain.FriendlyName` (.NET) during initialization. For Java, `Ice.ProgramName` is initialized to the empty string. The default name can be overridden by setting this property.

Ice.RetryIntervals

Synopsis

`Ice.RetryIntervals=`*num* `[`*num* `...]`

Description

This property defines the number of times an operation is automatically retried and the delay between each retry. For example, if the property is set to `0 100 500`, the operation is retried 3 times: immediately after the first failure, again after waiting 100ms after the second failure, and again after waiting 500ms after the third failure. The default value (`0`) means Ice retries once immediately. If set to `-1`, no retry occurs.

Ice.ServerIdleTime

Synopsis

`Ice.ServerIdleTime=`*num*

Description

If *num* is set to a value larger than zero, Ice automatically calls `Communicator::shutdown` once the communicator has been idle for *num* seconds. This shuts down the communicator's server side and causes all threads waiting in `Communicator::waitForShutdown` to return. After that, a server will typically do some clean-up work before exiting. The default value is zero, meaning that the server will not shut down automatically. This property is often used for servers that are automatically activated by IceGrid.

Ice.SOCKSProxyHost

Synopsis

`Ice.SOCKSProxyHost=`*addr*

Description

Specifies the host name or IP address of a SOCKS proxy server. If *addr* is not empty, Ice uses the designated SOCKS proxy server for all outgoing (client) connections.

Ice currently only supports the SOCKS4 protocol, which means only IPv4 connections are allowed.

Ice.SOCKSProxyPort

Synopsis

`Ice.SOCKSProxyPort=`*num*

Description

The port number of the SOCKS proxy server. If not specified, the default value is `1080`.

Ice.StdErr

Synopsis

`Ice.StdErr=`*filename*

Description

If *`filename`* is not empty, the standard error stream of this process is redirected to this file, in append mode. This property is checked only for the first communicator that is created in a process.

Ice.StdOut

Synopsis

`Ice.StdOut=`*`filename`*

Description

If *`filename`* is not empty, the standard output stream of this process is redirected to this file, in append mode. This property is checked only for the first communicator created in a process.

Ice.SyslogFacility

Synopsis

`Ice.SyslogFacility=`*`string`* (Unix only)

Description

This property sets the syslog facility to *`string`*. This property has no effect if `Ice.UseSyslog` is not set.

`string` can be any of syslog facilities: `LOG_AUTH`, `LOG_AUTHPRIV`, `LOG_CRON`, `LOG_DAEMON`, `LOG_FTP`, `LOG_KERN`, `LOG_LOCAL0`, `LOG_LOCAL1`, `LOG_LOCAL2`, `LOG_LOCAL3`, `LOG_LOCAL4`, `LOG_LOCAL5`, `LOG_LOCAL6`, `LOG_LOCAL7`, `LOG_LPR`, `LOG_MAIL`, `LOG_NEWS`, `LOG_SYSLOG`, `LOG_USER`, `LOG_UUCP`.

The default value is `LOG_USER`.

Ice.Voip

Synopsis

`Ice.Voip=`*`num`* (Ice Touch only)

Description

If *`num`* is set to a value larger than zero, the Ice run time sets the `kCFStreamNetworkServiceType` property to `kCFStreamNetworkServiceType VoIP` for all sockets created through the Ice communicator. Only voice over IP applications are allowed to use this setting. It ensures the sockets aren't closed and enables receiving data when the application is in the background. See the description from Configuring Sockets for VoIP Usage for information on this setting and when it's applicable.

The default value is zero.

Ice.UseSyslog

Synopsis

`Ice.UseSyslog=`*`num`* (Unix only)

Description

If *`num`* is set to a value larger than zero, a special logger is installed that logs to the `syslog` service instead of standard error. The identifier for `syslog` is the value of `#Ice.ProgramName`. Use `#Ice.SyslogFacility` to select a `syslog` facility.