Ice-Communicator

Ice::Communicator

Overview

local interface Communicator

The central object in Ice. One or more communicators can be instantiated for an Ice application. Communicator instantiation is language-specific, and not specified in Slice code.

Used By

- Freeze::Connection::getCommunicator
- Ice::ObjectAdapter::getCommunicator
- IceBox::Service::start

See Also

- Ice::Logger
- Ice::Stats
- Ice::ObjectAdapter
- Ice::Properties
- Ice::ObjectFactory

Operation Index

destroy — Destroy the communicator.
shutdown — Shuts down this communicator's server functionality, which includes the deactivation of all object adapters.
waitForShutdown — Wait until the application has called shutdown (or destroy).
isShutdown — Check whether communicator has been shut down.
stringToProxy — Convert a stringified proxy into a proxy.
proxyToString — Convert a proxy into a string.
propertyToProxy — Convert a set of proxy properties into a proxy.
proxyToProperty — Convert a proxy to a set of proxy properties.
stringToIdentity — Convert a string into an identity.
identityToString — Convert an identity into a string.
createObjectAdapter — Create a new object adapter.
createObjectAdapterWithEndpoints — Create a new object adapter with endpoints.
createObjectAdapterWithRouter — Create a new object adapter with a router.
addObjectFactory — Add an object factory to this communicator.
findObjectFactory — Find an object factory registered with this communicator.
getImplicitContext — Get the implicit context associated with this communicator.
getProperties — Get the properties for this communicator.
getLogger — Get the logger for this communicator.
getStats — Get the statistics callback object for this communicator.
getObserver — Get the observer resolver object for this communicator.
getDefaultRouter — Get the default router this communicator.
setDefaultRouter — Set a default router for this communicator.
getDefaultLocator — Get the default locator this communicator.
setDefaultLocator — Set a default Ice locator for this communicator.
getPluginManager — Get the plug-in manager for this communicator.
flushBatchRequests — Flush any pending batch requests for this communicator.
getAdmin — Get a proxy to the main facet of the Admin object.
addAdminFacet — Add a new facet to the Admin object.
removeAdminFacet — Remove the following facet to the Admin object.
findAdminFacet — Returns a facet of the Admin object.

Operations

void destroy()

Destroy the communicator. This operation calls shutdown implicitly. Calling destroy cleans up memory, and shuts down this communicator's client functionality and destroys all object adapters. Subsequent calls to destroy are ignored.

See Also

- shutdown
- Ice::ObjectAdapter::destroy

void shutdown()

Shuts down this communicator's server functionality, which includes the deactivation of all object adapters. (Attempts to use a deactivated object adapter raise Ice::ObjectAdapterDeactivatedException.) Subsequent calls to shutdown are ignored.

After shutdown returns, no new requests are processed. However, requests that have been started before shutdown was called might still be active. You can use waitForShutdown to wait for the completion of all requests.

See Also

- destroy
- waitForShutdown
- Ice::ObjectAdapter::deactivate

void waitForShutdown()

Wait until the application has called shutdown (or destroy). On the server side, this operation blocks the calling thread until all currently-executing operations have completed. On the client side, the operation simply block until another thread has called shutdown or destroy.

A typical use of this operation is to call it from the main thread, which then waits until some other thread calls shutdown. After shut-down is complete, the main thread returns and can do some cleanup work before it finally calls destroy to shut down the client functionality, and then exits the application.

See Also

- shutdown
- destroy
- Ice::ObjectAdapter::waitForDeactivate

bool isShutdown()

Check whether communicator has been shut down.

Return Value

True if the communicator has been shut down; false otherwise.

See Also

- shutdown

Object* stringToProxy(string str)

Convert a stringified proxy into a proxy. For example, `MyCategory/MyObject:tcp -h some_host -p 10000` creates a proxy that refers to the Ice object having an identity with a name "MyObject" and a category "MyCategory", with the server running on host "some_host", port 10000. If the stringified proxy does not parse correctly, the operation throws one of Ice::ProxyParseException, Ice::EndpointParseException, or Ice::IdentityParseException. An appendix in the Ice manual provides a detailed description of the syntax supported by stringified proxies.

Parameters

`str` — The stringified proxy to convert into a proxy.

Return Value

The proxy, or nil if `str` is an empty string.

See Also

- proxyToString

string proxyToString(Object* obj)

Convert a proxy into a string.

Parameters

`obj` — The proxy to convert into a stringified proxy.

Return Value

The stringified proxy, or an empty string if `obj` is nil.

See Also

- stringToProxy

Object* propertyToProxy(string property)

Convert a set of proxy properties into a proxy. The "base" name supplied in the `property` argument refers to a property containing a stringified proxy, such as `MyProxy=id:tcp -h localhost -p 10000`. Additional properties configure local settings for the proxy, such as `MyProxy.PreferSecure=1`. The "Properties" appendix in the Ice manual describes each of the supported proxy properties.

Parameters

`property` — The base property name.

Return Value

The proxy.

Ice::PropertyDict proxyToProperty(Object* proxy, string property)

Convert a proxy to a set of proxy properties.

Parameters

`proxy` — The proxy.
`property` — The base property name.

Return Value

The property set.

Ice::Identity stringToIdentity(string str)

Convert a string into an identity. If the string does not parse correctly, the operation throws Ice::IdentityParseException.

Parameters

`str` — The string to convert into an identity.

Return Value

The identity.

See Also

- identityToString

string identityToString(Ice::Identity ident)

Convert an identity into a string.

Parameters

`ident` — The identity to convert into a string.

Return Value

The "stringified" identity.

See Also

- stringToIdentity

Ice::ObjectAdapter createObjectAdapter(string name)

Create a new object adapter. The endpoints for the object adapter are taken from the property `name.Endpoints`.

It is legal to create an object adapter with the empty string as its name. Such an object adapter is accessible via bidirectional connections or by collocated invocations that originate from the same communicator as is used by the adapter.

Attempts to create a named object adapter for which no configuration can be found raise Ice::InitializationException.

Parameters

`name` — The object adapter name.

Return Value

The new object adapter.

See Also

- createObjectAdapterWithEndpoints
- Ice::ObjectAdapter
- Ice::Properties

Ice::ObjectAdapter createObjectAdapterWithEndpoints(string name, string endpoints)

Create a new object adapter with endpoints. This operation sets the property `name.Endpoints`, and then calls createObjectAdapter. It is provided as a convenience function.

Calling this operation with an empty name will result in a UUID being generated for the name.

Parameters

`name` — The object adapter name.
`endpoints` — The endpoints for the object adapter.

Return Value

The new object adapter.

See Also

- createObjectAdapter
- Ice::ObjectAdapter
- Ice::Properties

Ice::ObjectAdapter createObjectAdapterWithRouter(string name, Ice::Router* rtr)

Create a new object adapter with a router. This operation creates a routed object adapter.

Calling this operation with an empty name will result in a UUID being generated for the name.

Parameters

`name` — The object adapter name.
`rtr` — The router.

Return Value

The new object adapter.

See Also

- createObjectAdapter
- Ice::ObjectAdapter
- Ice::Properties

void addObjectFactory(Ice::ObjectFactory factory, string id)

Add an object factory to this communicator. Installing a factory with an id for which a factory is already registered throws Ice::AlreadyRegisteredException.

When unmarshaling an Ice object, the Ice run time reads the most-derived type id off the wire and attempts to create an instance of the type using a factory. If no instance is created, either because no factory was found, or because all factories returned nil, the behavior of the Ice run time depends on the format with which the object was marshaled:

If the object uses the "sliced" format, Ice ascends the class hierarchy until it finds a type that is recognized by a factory, or it reaches the least-derived type. If no factory is found that can create an instance, the run time throws Ice::NoObjectFactoryException.

If the object uses the "compact" format, Ice immediately raises Ice::NoObjectFactoryException.

The following order is used to locate a factory for a type:

1. The Ice run-time looks for a factory registered specifically for the type.
2. If no instance has been created, the Ice run-time looks for the default factory, which is registered with an empty type id.
3. If no instance has been created by any of the preceding steps, the Ice run-time looks for a factory that may have been statically generated by the language mapping for non-abstract classes.

Parameters

`factory` — The factory to add.
`id` — The type id for which the factory can create instances, or an empty string for the default factory.

See Also

- findObjectFactory
- Ice::ObjectFactory

Ice::ObjectFactory findObjectFactory(string id)

Find an object factory registered with this communicator.

Parameters

`id` — The type id for which the factory can create instances, or an empty string for the default factory.

Return Value

The object factory, or null if no object factory was found for the given id.

See Also

- addObjectFactory
- Ice::ObjectFactory

Ice::ImplicitContext getImplicitContext()

Get the implicit context associated with this communicator.

Return Value

The implicit context associated with this communicator; returns null when the property Ice.ImplicitContext is not set or is set to None.

Ice::Properties getProperties()

Get the properties for this communicator.

Return Value

This communicator's properties.

See Also

- Ice::Properties

Ice::Logger getLogger()

Get the logger for this communicator.

Return Value

This communicator's logger.

See Also

- Ice::Logger

Ice::Stats getStats()

Get the statistics callback object for this communicator.

Return Value

This communicator's statistics callback object.

See Also

- Ice::Stats

Ice::Instrumentation::CommunicatorObserver getObserver()

Get the observer resolver object for this communicator.

Return Value

This communicator's observer resolver object.

See Also

- Ice::Stats

Ice::Router* getDefaultRouter()

Get the default router this communicator.

Return Value

The default router for this communicator.

See Also

- setDefaultRouter
- Ice::Router

void setDefaultRouter(Ice::Router* rtr)

Set a default router for this communicator. All newly created proxies will use this default router. To disable the default router, null can be used. Note that this operation has no effect on existing proxies.

You can also set a router for an individual proxy by calling the operation `ice_router` on the proxy.

Parameters

`rtr` — The default router to use for this communicator.

See Also

- getDefaultRouter
- createObjectAdapterWithRouter
- Ice::Router

Ice::Locator* getDefaultLocator()

Get the default locator this communicator.

Return Value

The default locator for this communicator.

See Also

- setDefaultLocator
- Ice::Locator

void setDefaultLocator(Ice::Locator* loc)

Set a default Ice locator for this communicator. All newly created proxy and object adapters will use this default locator. To disable the default locator, null can be used. Note that this operation has no effect on existing proxies or object adapters.

You can also set a locator for an individual proxy by calling the operation `ice_locator` on the proxy, or for an object adapter by calling the operation `setLocator` on the object adapter.

Parameters

`loc` — The default locator to use for this communicator.

See Also

- getDefaultLocator
- Ice::Locator
- Ice::ObjectAdapter::setLocator

Ice::PluginManager getPluginManager()

Get the plug-in manager for this communicator.

Return Value

This communicator's plug-in manager.

See Also

- Ice::PluginManager

["async"] void flushBatchRequests()

Flush any pending batch requests for this communicator. This causes all batch requests that were sent via proxies obtained via this communicator to be sent to the server.

Object* getAdmin()

Get a proxy to the main facet of the Admin object. When Ice.Admin.DelayCreation is greater than 0, it is necessary to call getAdmin() after the communicator is initialized to create the Admin object. Otherwise, the Admin object is created automatically after all the plug-ins are initialized.

Return Value

The main ("") facet of the Admin object; a null proxy if no Admin object is configured.

void addAdminFacet(Object servant, string facet)

Add a new facet to the Admin object. Adding a servant with a facet that is already registered throws Ice::AlreadyRegisteredException.

Parameters

`servant` — The servant that implements the new Admin facet.
`facet` — The name of the new Admin facet.

Object removeAdminFacet(string facet)

Remove the following facet to the Admin object. Removing a facet that was not previously registered throws Ice::NotRegisteredException.

Parameters

`facet` — The name of the Admin facet.

Return Value

The servant associated with this Admin facet.

Object findAdminFacet(string facet)

Returns a facet of the Admin object.

Parameters

`facet` — The name of the Admin facet.

Return Value

The servant associated with this Admin facet, or null if no facet is registered with the given name.