Ice-ObjectAdapter

Ice::ObjectAdapter

Overview

local interface ObjectAdapter

The object adapter provides an up-call interface from the Ice run time to the implementation of Ice objects. The object adapter is responsible for receiving requests from endpoints, and for mapping between servants, identities, and proxies.

Used By

- Freeze::ServantInitializer::initialize
- Ice::Communicator::createObjectAdapterWithEndpoints
- Ice::Communicator::createObjectAdapterWithRouter
- Ice::Communicator::createObjectAdapter
- Ice::Connection::getAdapter
- Ice::Connection::setAdapter
- Ice::Current::adapter

See Also

- Ice::Communicator
- Ice::ServantLocator

Operation Index

Operations

string getName()

Get the name of this object adapter.

Return Value

This object adapter's name.

Ice::Communicator getCommunicator()

Get the communicator this object adapter belongs to.

Return Value

This object adapter's communicator.

See Also

- Ice::Communicator

void activate()

Activate all endpoints that belong to this object adapter. After activation, the object adapter can dispatch requests received through its endpoints.

See Also

- hold
- deactivate

void hold()

Temporarily hold receiving and dispatching requests. The object adapter can be reactivated with the activate operation.

Holding is not immediate, i.e., after hold returns, the object adapter might still be active for some time. You can use waitForHold to wait until holding is complete.

See Also

- activate
- deactivate
- waitForHold

void waitForHold()

Wait until the object adapter holds requests. Calling hold initiates holding of requests, and waitForHold only returns when holding of requests has been completed.

See Also

- hold
- waitForDeactivate
- Ice::Communicator::waitForShutdown

void deactivate()

Deactivate all endpoints that belong to this object adapter. After deactivation, the object adapter stops receiving requests through its endpoints. Object adapters that have been deactivated must not be reactivated again, and cannot be used otherwise. Attempts to use a deactivated object adapter raise Ice::ObjectAdapterDeactivatedException; however, attempts to deactivate an already deactivated object adapter are ignored and do nothing. Once deactivated, it is possible to destroy the adapter to clean up resources and then create and activate a new adapter with the same name.

After deactivate returns, no new requests are processed by the object adapter. However, requests that have been started before deactivate was called might still be active. You can use waitForDeactivate to wait for the completion of all requests for this object adapter.

See Also

- activate
- hold
- waitForDeactivate
- Ice::Communicator::shutdown

void waitForDeactivate()

Wait until the object adapter has deactivated. Calling deactivate initiates object adapter deactivation, and waitForDeactivate only returns when deactivation has been completed.

See Also

- deactivate
- waitForHold
- Ice::Communicator::waitForShutdown

bool isDeactivated()

Check whether object adapter has been deactivated.

Return Value

Whether adapter has been deactivated.

See Also

- Ice::Communicator::shutdown

void destroy()

Destroys the object adapter and cleans up all resources held by the object adapter. If the object adapter has not yet been deactivated, destroy implicitly initiates the deactivation and waits for it to finish. Subsequent calls to destroy are ignored. Once destroy has returned, it is possible to create another object adapter with the same name.

See Also

- deactivate
- waitForDeactivate
- Ice::Communicator::destroy

Object* add(Object servant, Ice::Identity id)

Add a servant to this object adapter's Active Servant Map. Note that one servant can implement several Ice objects by registering the servant with multiple identities. Adding a servant with an identity that is in the map already throws Ice::AlreadyRegisteredException.

Parameters

`servant` — The servant to add.
`id` — The identity of the Ice object that is implemented by the servant.

Return Value

A proxy that matches the given identity and this object adapter.

See Also

- Ice::Identity
- addFacet
- addWithUUID
- remove
- find

Object* addFacet(Object servant, Ice::Identity id, string facet)

Like add, but with a facet. Calling `add(servant, id)` is equivalent to calling addFacet with an empty facet.

Parameters

`servant` — The servant to add.
`id` — The identity of the Ice object that is implemented by the servant.
`facet` — The facet. An empty facet means the default facet.

Return Value

A proxy that matches the given identity, facet, and this object adapter.

See Also

- Ice::Identity
- add
- addFacetWithUUID
- removeFacet
- findFacet

Object* addWithUUID(Object servant)

Add a servant to this object adapter's Active Servant Map, using an automatically generated UUID as its identity. Note that the generated UUID identity can be accessed using the proxy's `ice_getIdentity` operation.

Parameters

`servant` — The servant to add.

Return Value

A proxy that matches the generated UUID identity and this object adapter.

See Also

- Ice::Identity
- add
- addFacetWithUUID
- remove
- find

Object* addFacetWithUUID(Object servant, string facet)

Like addWithUUID, but with a facet. Calling `addWithUUID(servant)` is equivalent to calling addFacetWithUUID with an empty facet.

Parameters

`servant` — The servant to add.
`facet` — The facet. An empty facet means the default facet.

Return Value

A proxy that matches the generated UUID identity, facet, and this object adapter.

See Also

- Ice::Identity
- addFacet
- addWithUUID
- removeFacet
- findFacet

void addDefaultServant(Object servant, string category)

Add a default servant to handle requests for a specific category. Adding a default servant for a category for which a default servant is already registered throws Ice::AlreadyRegisteredException. To dispatch operation calls on servants, the object adapter tries to find a servant for a given Ice object identity and facet in the following order:

1. The object adapter tries to find a servant for the identity and facet in the Active Servant Map.
2. If no servant has been found in the Active Servant Map, the object adapter tries to find a default servant for the category component of the identity.
3. If no servant has been found by any of the preceding steps, the object adapter tries to find a default servant for an empty category, regardless of the category contained in the identity.
4. If no servant has been found by any of the preceding steps, the object adapter gives up and the caller receives Ice::ObjectNotExistException or Ice::FacetNotExistException.

Parameters

`servant` — The default servant.
`category` — The category for which the default servant is registered. An empty category means it will handle all categories.

See Also

- removeDefaultServant
- findDefaultServant

Object remove(Ice::Identity id)

Remove a servant (that is, the default facet) from the object adapter's Active Servant Map.

Parameters

`id` — The identity of the Ice object that is implemented by the servant. If the servant implements multiple Ice objects, remove has to be called for all those Ice objects. Removing an identity that is not in the map throws Ice::NotRegisteredException.

Return Value

The removed servant.

See Also

- Ice::Identity
- add
- addWithUUID

Object removeFacet(Ice::Identity id, string facet)

Like remove, but with a facet. Calling `remove(id)` is equivalent to calling removeFacet with an empty facet.

Parameters

`id` — The identity of the Ice object that is implemented by the servant.
`facet` — The facet. An empty facet means the default facet.

Return Value

The removed servant.

See Also

- Ice::Identity
- addFacet
- addFacetWithUUID

Ice::FacetMap removeAllFacets(Ice::Identity id)

Remove all facets with the given identity from the Active Servant Map. The operation completely removes the Ice object, including its default facet. Removing an identity that is not in the map throws Ice::NotRegisteredException.

Parameters

`id` — The identity of the Ice object to be removed.

Return Value

A collection containing all the facet names and servants of the removed Ice object.

See Also

- remove
- removeFacet

Object removeDefaultServant(string category)

Remove the default servant for a specific category. Attempting to remove a default servant for a category that is not registered throws Ice::NotRegisteredException.

Parameters

`category` — The category of the default servant to remove.

Return Value

The default servant.

See Also

- addDefaultServant

- findDefaultServant

Object find(Ice::Identity id)

Look up a servant in this object adapter's Active Servant Map by the identity of the Ice object it implements.

This operation only tries to look up a servant in the Active Servant Map. It does not attempt to find a servant by using any installed Ice::ServantLocator

Parameters

`id` — The identity of the Ice object for which the servant should be returned.

Return Value

The servant that implements the Ice object with the given identity, or null if no such servant has been found.

See Also

- Ice::Identity
- findFacet
- findByProxy

Object findFacet(Ice::Identity id, string facet)

Like find, but with a facet. Calling `find(id)` is equivalent to calling findFacet with an empty facet.

Parameters

`id` — The identity of the Ice object for which the servant should be returned.
`facet` — The facet. An empty facet means the default facet.

Return Value

The servant that implements the Ice object with the given identity and facet, or null if no such servant has been found.

See Also

- Ice::Identity
- find
- findByProxy

Ice::FacetMap findAllFacets(Ice::Identity id)

Find all facets with the given identity in the Active Servant Map.

Parameters

`id` — The identity of the Ice object for which the facets should be returned.

Return Value

A collection containing all the facet names and servants that have been found, or an empty map if there is no facet for the given identity.

See Also

- find
- findFacet

Object findByProxy(Object* proxy)

Look up a servant in this object adapter's Active Servant Map, given a proxy.

This operation only tries to lookup a servant in the Active Servant Map. It does not attempt to find a servant by using any installed Ice::ServantLocator.

Parameters

`proxy` — The proxy for which the servant should be returned.

Return Value

The servant that matches the proxy, or null if no such servant has been found.

See Also

- find
- findFacet

void addServantLocator(Ice::ServantLocator locator, string category)

Add a Servant Locator to this object adapter. Adding a servant locator for a category for which a servant locator is already registered throws Ice::AlreadyRegisteredException. To dispatch operation calls on servants, the object adapter tries to find a servant for a given Ice object identity and facet in the following order:

1. The object adapter tries to find a servant for the identity and facet in the Active Servant Map.
2. If no servant has been found in the Active Servant Map, the object adapter tries to find a servant locator for the category component of the identity. If a locator is found, the object adapter tries to find a servant using this locator.
3. If no servant has been found by any of the preceding steps, the object adapter tries to find a locator for an empty category, regardless of the category contained in the identity. If a locator is found, the object adapter tries to find a servant using this locator.
4. If no servant has been found by any of the preceding steps, the object adapter gives up and the caller receives Ice::ObjectNotExistException or Ice::FacetNotExistException.

Only one locator for the empty category can be installed.

Parameters

`locator` — The locator to add.
`category` — The category for which the Servant Locator can locate servants, or an empty string if the Servant Locator does not belong to any specific category.

See Also

- Ice::Identity
- removeServantLocator
- findServantLocator
- Ice::ServantLocator

Ice::ServantLocator removeServantLocator(string category)

Remove a Servant Locator from this object adapter.

Parameters

`category` — The category for which the Servant Locator can locate servants, or an empty string if the Servant Locator does not belong to any specific category.

Return Value

The Servant Locator, or throws Ice::NotRegisteredException if no Servant Locator was found for the given category.

See Also

- Ice::Identity
- addServantLocator
- findServantLocator
- Ice::ServantLocator

Ice::ServantLocator findServantLocator(string category)

Find a Servant Locator installed with this object adapter.

Parameters

`category` — The category for which the Servant Locator can locate servants, or an empty string if the Servant Locator does not belong to any specific category.

Return Value

The Servant Locator, or null if no Servant Locator was found for the given category.

See Also

- Ice::Identity
- addServantLocator
- removeServantLocator
- Ice::ServantLocator

Object findDefaultServant(string category)

Find the default servant for a specific category.

Parameters

`category` — The category of the default servant to find.

Return Value

The default servant or null if no default servant was registered for the category.

See Also

- addDefaultServant
- removeDefaultServant

Object* createProxy(Ice::Identity id)

Create a proxy for the object with the given identity. If this object adapter is configured with an adapter id, the return value is an indirect proxy that refers to the adapter id. If a replica group id is also defined, the return value is an indirect proxy that refers to the replica group id. Otherwise, if no adapter id is defined, the return value is a direct proxy containing this object adapter's published endpoints.

Parameters

`id` — The object's identity.

Return Value

A proxy for the object with the given identity.

See Also

- Ice::Identity

Object* createDirectProxy(Ice::Identity id)

Create a direct proxy for the object with the given identity. The returned proxy contains this object adapter's published endpoints.

Parameters

`id` — The object's identity.

Return Value

A proxy for the object with the given identity.

See Also

- Ice::Identity

Object* createIndirectProxy(Ice::Identity id)

Create an indirect proxy for the object with the given identity. If this object adapter is configured with an adapter id, the return value refers to the adapter id. Otherwise, the return value contains only the object identity.

Parameters

`id` — The object's identity.

Return Value

A proxy for the object with the given identity.

See Also

- Ice::Identity

void setLocator(Ice::Locator* loc)

Set an Ice locator for this object adapter. By doing so, the object adapter will register itself with the locator registry when it is activated for the first time. Furthermore, the proxies created by this object adapter will contain the adapter name instead of its endpoints.

Parameters

`loc` — The locator used by this object adapter.

See Also

- createDirectProxy
- Ice::Locator
- Ice::LocatorRegistry

void refreshPublishedEndpoints()

Refresh the set of published endpoints. The run time re-reads the PublishedEndpoints property if it is set and re-reads the list of local interfaces if the adapter is configured to listen on all endpoints. This operation is useful to refresh the endpoint information that is published in the proxies that are created by an object adapter if the network interfaces used by a host changes.

Ice::EndpointSeq getEndpoints()

Get the set of endpoints configured with this object adapter.

Return Value

The set of endpoints.

See Also

- Ice::Endpoint

Ice::EndpointSeq getPublishedEndpoints()

Get the set of endpoints that proxies created by this object adapter will contain.

Return Value

The set of published endpoints.

See Also

- refreshPublishedEndpoints
- Ice::Endpoint