Proxy Methods

Although the core proxy functionality is supplied by a language-specific base class, we can describe the proxy methods in terms of Slice operations as shown below:

Slice

```
bool ice_isA(string id);
void ice_ping();
StringSeq ice_ids();
string ice_id();
Communicator ice_getCommunicator();
string ice_toString();
Object* ice_identity(Identity id);
Identity ice_getIdentity();
Object* ice_adapterId(string id);
string ice_getAdapterId();
Object* ice_endpoints(EndpointSeq endpoints);
EndpointSeq ice_getEndpoints();
Object* ice_endpointSelection(EndpointSelectionType t);
EndpointSelectionType ice_getEndpointSelection();
Object* ice_context(Context ctx);
Context ice_getContext();
Object* ice_facet(string facet);
string ice_getFacet();
Object* ice_twoway();
bool ice_isTwoway();
Object* ice_oneway();
bool ice_isOneway();
Object* ice_batchOneway();
bool ice_isBatchOneway();
Object* ice_datagram();
bool ice_isDatagram();
Object* ice_batchDatagram();
bool ice_isBatchDatagram();
Object* ice_secure(bool b);
bool ice_isSecure();
EncodingVersion ice_getEncodingVersion();
Object* ice_encodingVersion(EncodingVersion v);
Object* ice_preferSecure(bool b);
bool ice_isPreferSecure();
Object* ice_compress(bool b);
Object* ice_timeout(int timeout);
Object* ice_router(Router* rtr);
Router* ice_getRouter();
Object* ice_locator(Locator* loc);
Locator* ice_getLocator();
Object* ice_locatorCacheTimeout(int seconds);
int ice_getLocatorCacheTimeout();
Object* ice_collocationOptimized(bool b);
bool ice_isCollocationOptimized();
Object* ice_invocationTimeout(int timeout);
int ice_getInvocationTimeout();
Object* ice_connectionId(string id);
string ice_getConnectionId();
Connection ice_getConnection();
Connection ice_getCachedConnection();
Object* ice_connectionCached(bool b);
bool ice_isConnectionCached();
void ice_flushBatchRequests();
bool ice_invoke(string operation, OperationMode mode,
                ByteSeq inParams, out ByteSeq outParams);

// Only with C# mapping
System.Threading.Tasks.TaskScheduler ice_scheduler();
// Only with Java mapping
java.util.concurrent.Executor ice_executor();
```

These methods can be categorized as follows:

- Remote inspection: methods that return information about the remote object. These methods make remote invocations and therefore accept an optional trailing argument of type `Ice::Context`.
- Local inspection: methods that return information about the proxy's local configuration.
- Factory: methods that return new proxy instances configured with different features.

- Request processing: methods that flush batch requests and send "dynamic" Ice invocations.

Proxies are immutable, so factory methods allow an application to obtain a new proxy with the desired configuration. Factory methods essentially clone the original proxy and modify one or more features of the new proxy.

Many of the factory methods are not supported by fixed proxies, which are used in conjunction with bidirectional connections. Attempting to invoke one of these methods causes the Ice run time to raise `FixedProxyException`.

The core proxy methods are explained in greater detail in the following table:

Method	Description	Remote
`ice_isA`	Returns `true` if the remote object supports the type indicated by the `id` argument, otherwise `false`. This method can only be invoked on a twoway proxy.	Yes
`ice_ping`	Determines whether the remote object is reachable. Does not return a value.	Yes
`ice_ids`	Returns the type IDs of the types supported by the remote object. The return value is an array of strings. This method can only be invoked on a twoway proxy.	Yes
`ice_id`	Returns the type ID of the most-derived type supported by the remote object. This method can only be invoked on a twoway proxy.	Yes
`ice_getCommunicator`	Returns the communicator that was used to create this proxy.	No
`ice_toString`	Returns the string representation of the proxy. The `Ice.ToStringMode` property controls how non-printable ASCII characters and non-ASCII characters in the identity, facet and object adapter id of this proxy are represented in the resulting string. When called on a fixed proxy, this method returns a stringified proxy without endpoints.	No
`ice_identity`	Returns a new proxy having the given identity.	No
`ice_getIdentity`	Returns the identity of the Ice object represented by the proxy.	No
`ice_adapterId`	Returns a new proxy having the given adapter ID.	No
`ice_getAdapterId`	Returns the proxy's adapter ID, or an empty string if no adapter ID is configured (as is the case with direct proxies).	No
`ice_endpoints`	Returns a new proxy having the given endpoints.	No
`ice_getEndpoints`	Returns a sequence of `Endpoint` objects representing the direct proxy's endpoints. For indirect proxies, an empty sequence is returned.	No
`ice_endpointSelection`	Returns a new proxy having the given endpoint selection policy (random or ordered).	No
`ice_getEndpointSelection`	Returns the endpoint selection policy for the proxy.	No
`ice_context`	Returns a new proxy having the given request context.	No
`ice_getContext`	Returns the request context associated with the proxy.	No
`ice_facet`	Returns a new proxy having the given facet name.	No
`ice_getFacet`	Returns the name of the facet associated with the proxy, or an empty string if no facet has been set.	No
`ice_twoway`	Returns a new proxy for making twoway invocations.	No
`ice_isTwoway`	Returns `true` if the proxy uses twoway invocations, otherwise `false`.	No

`ice_oneway`	Returns a new proxy for making oneway invocations.	No
`ice_isOneway`	Returns `true` if the proxy uses oneway invocations, otherwise `false`.	No
`ice_batchOneway`	Returns a new proxy for making batch oneway invocations.	No
`ice_isBatchOneway`	Returns `true` if the proxy uses batch oneway invocations, otherwise `false`.	No
`ice_datagram`	Returns a new proxy for making datagram invocations.	No
`ice_isDatagram`	Returns `true` if the proxy uses datagram invocations, otherwise `false`.	No
`ice_batchDatagram`	Returns a new proxy for making batch datagram invocations.	No
`ice_isBatchDatagram`	Returns `true` if the proxy uses batch datagram invocations, otherwise `false`.	No
`ice_secure`	Returns a new proxy whose endpoints may be filtered depending on the boolean argument. If `true`, only endpoints using secure transports are allowed, otherwise all endpoints are allowed.	No
`ice_getEncodingVersion`	Returns the encoding version that is used to encode requests invoked on this proxy.	No
`ice_encodingVersion`	Returns a new proxy that uses the given encoding version to encode requests. Raises `UnsupportedEncodingException` if the version is invalid.	No
`ice_isSecure`	Returns `true` if the proxy uses only secure endpoints, otherwise `false`.	No
`ice_preferSecure`	Returns a new proxy whose endpoints are filtered depending on the boolean argument. If `true`, endpoints using secure transports are given precedence over endpoints using non-secure transports. If `false`, the default behavior gives precedence to endpoints using non-secure transports.	No
`ice_isPreferSecure`	Returns `true` if the proxy prefers secure endpoints, otherwise `false`.	No
`ice_compress`	Returns a new proxy whose protocol compression capability is determined by the boolean argument. If `true`, the proxy uses protocol compression if it is supported by the endpoint. If `false`, protocol compression is never used.	No
`ice_timeout`	Returns a new proxy whose endpoints all have the given connection timeout value in milliseconds. A value of `-1` disables timeouts.	No
`ice_router`	Returns a new proxy configured with the given router proxy.	No
`ice_getRouter`	Returns the router that is configured for the proxy (null if no router is configured).	No
`ice_locator`	Returns a new proxy with the specified locator.	No
`ice_getLocator`	Returns the locator that is configured for the proxy (null if no locator is configured).	No
`ice_locatorCacheTimeout`	Returns a new proxy with the specified locator cache timeout in seconds. When binding a proxy to an endpoint, the run time caches the proxy returned by the locator and uses the cached proxy while the cached proxy has been in the cache for less than the timeout. Proxies older than the timeout cause the run time to rebind via the locator. A value of 0 disables caching entirely, and a value of -1 means that cached proxies never expire. The default value is -1.	No
`ice_getLocatorCacheTimeout`	Returns the locator cache timeout value in seconds.	No

`ice_collocationOptimized`	Returns a new proxy configured for collocation optimization. If `true`, collocated optimizations are enabled. The default value is `true`.	No
`ice_isCollocationOptimized`	Returns `true` if the proxy uses collocation optimization, otherwise `false`.	No
`ice_invocationTimeout`	Returns a new proxy configured with the specified invocation timeout in milliseconds. A value of `-1` means an invocation never times out. A value of `-2` provides backward compatibility with Ice versions prior to 3.6 by disabling invocation timeouts and using connection timeouts to wait for the response of an invocation.	No
`ice_getInvocationTimeout`	Returns the invocation timeout value in milliseconds.	No
`ice_connectionId`	Returns a new proxy having the given connection ID.	No
`ice_getConnectionId`	Returns the connection ID, or an empty string if no connection ID has been configured.	No
`ice_getConnection` `ice_getConnectionAsync`	Returns an object representing the connection used by the proxy. If the proxy is not currently associated with a connection, the Ice run time attempts to establish a connection first, which means this method can potentially block while network operations are in progress. Use the asynchronous method to avoid blocking.	No
`ice_getCachedConnection`	Returns an object representing the connection used by the proxy, or null if the proxy is not currently associated with a connection.	No
`ice_connectionCached`	Enables or disables connection caching for the proxy.	No
`ice_isConnectionCached`	Returns `true` if the proxy uses connection caching, otherwise `false`.	No
`ice_flushBatchRequests`	Sends a batch of operation invocations synchronously or asynchronously.	Yes
`ice_invoke`	Allows dynamic invocation of an operation without the need for compiled Slice definitions. Requests can be sent synchronously or asynchronously.	Yes
`ice_scheduler`	This method is only available in the C# mapping. It returns a System.Threading.Tasks.TaskScheduler object that uses the Ice thread pool to execute tasks.	No
`ice_executor`	This method is only available in the Java mapping. It returns a java.util.concurrent.Executor object that uses the Ice thread pool to execute tasks.	No

Back to Top ^

See Also

- Request Contexts
- Oneway Invocations
- Batched Invocations
- Versioning
- Connection Timeouts
- Connection Establishment
- Using Connections
- Dynamic Invocation and Dispatch
- Asynchronous Dynamic Invocation and Dispatch
- Bidirectional Connections
- Glacier2
- Data Encoding

Previous

Next