IceDiscovery

IceDiscovery provides a location service using UDP multicast that enables Ice applications to discover objects and object adapters.

On this page:

IceDiscovery Overview

IceDiscovery is an Ice plug-in that must be installed in all of the clients and servers in your application. Once installed, IceDiscovery enables a client to dynamically locate objects using indirect proxies, which avoids the need for the client to statically configure the endpoints of the objects it uses. In a server, IceDiscovery makes objects and object adapters available for discovery with minimal additional effort.

IceDiscovery Concepts

This section reviews some concepts that will help you as you learn more about IceDiscovery.

Indirect Proxies

Indirect proxies have two formats:

- `identityOnly`
  This format (well-known proxy) uses only the object's identity.

- `identity@adapterId`
  This format combines the object identity and an adapter identifier. This identifier can either refer to a specific object adapter or a replica group.

Notice that neither format includes any endpoints, such as `tcp -h somehost -p 10000`. The ability to resolve an indirect proxy using only symbolic information, much like a DNS lookup, helps to loosen the coupling between clients and servers.

The Ice core delegates the resolution of indirect proxies to a standardized locator facility. This architecture offers a significant advantage: Ice applications can change locator settings via external configuration without requiring any changes to the application code.

Back to Top ^

Plug-in Facility

In addition to the locator facility, Ice also defines a standard plug-in facility through which applications can modify the functionality of the Ice core or add new capabilities, again using only external configuration.

IceDiscovery combines these two facilities: it's installed as a standard Ice plug-in via configuration, and this plug-in installs a custom locator implementation that enables discovery using UDP multicast.

We'll describe later how to install IceDiscovery in your clients and servers.

Back to Top ^

Replication

The locator facility includes support for replicated object adapters. For example, if `Adapter1` and `Adapter2` both participate in a replicate group identified as `TheGroup`, then the indirect proxy `someObject@TheGroup` could resolve to either `someObject@Adapter1` or `someObject@Adapter2`. Although there are two adapters that host `someObject` at the implementation level, both object adapters incarnate the same logical object. The application is responsible for ensuring that any persistent state is properly synchronized between the servers that host replicated object adapters.

Back to Top ^

IceDiscovery Domains

Nothing prevents two unrelated IceDiscovery applications from using the same multicast address and port, which means a plug-in from Application A can receive lookup requests from a client in Application B for objects and object adapters that might coincidentally match those of Application A. To avoid this situation, the applications should configure unique *domain identifiers*. The client plug-in includes its domain identifier in each lookup request it sends so that a server plug-in can ignore any requests that don't match its own domain identifier.

Back to Top ^

Discovery Process

When a client uses an indirect proxy with an adapter ID for the first time:

- The Ice run time queries the Ice locator implemented by the IceDiscovery plug-in to resolve the indirect proxy's adapter ID.
- The client's IceDiscovery plug-in transparently broadcasts a `findAdapterById` request via multicast.
- The lookup request includes the adapter ID, and a proxy that the target server's IceDiscovery plug-in can use to communicate directly with the client's IceDiscovery plug-in.
- Every server that installs the plug-in and uses the same addressing information receives the client's multicast lookup request; only the server that hosts the target adapter sends a reply.
- The client plug-in waits for a reply using a configurable timeout period; if it doesn't receive a reply, it tries again a configurable number of times before giving up.
- The target server's IceDiscovery plug-in sends a reply including a template proxy for the target adapter and whether or not it's replicated.
- The client's plug-in receives the reply and:
  - if the adapter is not replicated, it returns the proxy to the Ice run time location facility.
  - if the adapter is replicated, it waits again for replies from other servers. The duration of the wait is based on the time it took to receive the first reply (the latency) and a configurable latency multiplier.
- The Ice run time location facility caches the endpoints for the object adapter and the client's application code uses these cached endpoints to communicate directly with the target object using whichever transports its object adapter supports.

When the client uses a well-known proxy for the first time, an additional step occurs:

- The Ice run time queries the Ice locator implemented by the IceDiscovery plug-in to resolve the well-known proxy.
- The client's IceDiscovery plug-in transparently broadcasts a `findObjectById` request via multicast.
- The lookup request includes the identity of the target object, and a proxy that the target server's IceDiscovery plug-in can use to communicate directly with the client's IceDiscovery plug-in.
- Every server that installs the plug-in and uses the same addressing information receives the client's multicast lookup request; only the server that hosts this object sends a reply.
  - To check if a server hosts this object, the IceDiscovery plugin in the server searches the object adapters that have registered with the IceDiscovery LocatorRegistry by attempting to ping the object in these object adapters with `ice_ping`. It firsts checks the object adapters registered with a replica group ID, and then the object adapters registered without a replica group ID. This way, the object may be incarnated by a servant in the Active Servant Map, or by a default servant, or by a servant returned by a servant locator.
- The client plug-in waits for a reply using a configurable timeout period; if it doesn't receive a reply, it tries again a configurable number of times before giving up.
- The target server's IceDiscovery plug-in sends a reply including an indirect proxy for the target object.
- The Ice runtime location facility caches the indirect proxy for the well-known object.
- The indirect proxy is resolved with IceDiscovery using the steps mentioned above for indirect proxies.

Unless the Ice locator cache is disabled, only the initial lookup request occurs over multicast. Further requests use the information from the Ice runtime locator cache. The reply from the server plug-in to the client plug-in occurs using UDP unicast (by default). All subsequent communication between the client and the target object proceed directly without intervention by the IceDiscovery plug-in.

Back to Top ^

IceDiscovery vs. IceGrid

IceDiscovery and IceGrid both provide a location service but it helps to understand their differences when deciding which one to use in an application. Use IceDiscovery when your application needs a lightweight, transient location service. IceGrid's location service is backed by a persistent database and represents just one of the features that IceGrid offers, along with remote administration, on-demand server activation, and many others. If you need a location service but aren't yet ready to dive into IceGrid, start out using IceDiscovery; migrating to IceGrid later won't be difficult.

We provide a plug-in similar to IceDiscovery called IceLocatorDiscovery that integrates with IceGrid.

Back to Top ^

Installing IceDiscovery

The IceDiscovery plug-in must be installed in every client that needs to locate objects and in all of the servers that host those objects.

You can use the `Ice.Plugin` property to load and install the plug-in at runtime; the property value depends on the language mapping you're using:

C++

```
Ice.Plugin.IceDiscovery=IceDiscovery:createIceDiscovery
```

Java

```
Ice.Plugin.IceDiscovery=IceDiscovery:com.zeroc.IceDiscovery.PluginFactory
```

Java Compat

```
Ice.Plugin.IceDiscovery=IceDiscovery:IceDiscovery.PluginFactory
```

C#

```
Ice.Plugin.IceDiscovery=IceDiscovery:IceDiscovery.PluginFactory
```

Python

```
# Uses the C++ plug-in
Ice.Plugin.IceDiscovery=IceDiscovery:createIceDiscovery
```

Ruby

```
# Uses the C++ plug-in
Ice.Plugin.IceDiscovery=IceDiscovery:createIceDiscovery
```

PHP

```
# Uses the C++ plug-in
Ice.Plugin.IceDiscovery=IceDiscovery:createIceDiscovery
```

The C++ configuration is the same for the C++11 mapping and the C++98 mapping: Ice computes the name of the shared library to load and adds automatically a "++11" suffix when needed.

If using C++, instead of dynamically loading the plug-in at run time your application can explicitly link with and register the plug-in. To register the plug-in, you must call the `Ice::registerIceDiscovery(bool loadOnInitialize = true)` function before the communicator initialization. The `loadOnInitialize` parameter specifies if the plug-in is installed when the communicator is initialized. If set to `false`, you will need to enable the plugin by setting the `Ice.Plugin.IceDiscovery` property to `1`.

> `Ice::registerIceDiscovery` is a simple helper function that calls `Ice::registerPluginFactory`.

Refer to the next section for information on configuring the plug-in.

Back to Top ^

Configuring IceDiscovery

Applications configure the IceDiscovery plug-in using configuration properties; the plug-in does not provide a local API.

IceDiscovery Property Overview

The IceDiscovery plug-in supports a number of configuration properties, most of which affect the endpoints that the plug-in uses to communicate with its peers:

- Lookup endpoint
  This is the multicast endpoint on which all lookup queries are broadcast. It must use an IPv4 or IPv6 address in the multicast range with a fixed port.
- Reply endpoint
  This is the endpoint on which a client receives replies from servers. In general, this endpoint should not use a fixed port.

The plug-in uses sensible default values for all of its configuration properties, such that it's often unnecessary to define any of the plug-in's properties. However, it's still important to understand how the plug-in derives its endpoint information.

IceDiscovery creates several object adapters in each communicator in which it's installed, including the object adapters `IceDiscovery.Multicast` and `IceDiscovery.Reply`. These object adapters correspond to the Lookup and Reply endpoints mentioned above, respectively. You can configure the endpoints of these object adapters directly by defining the properties `IceDiscovery.Multicast.Endpoints` and `IceDiscovery.Reply.Endpoints`. If you don't define an endpoint for an object adapter, the plug-in computes it as follows:

- `IceDiscovery.Multicast.Endpoints=udp -h` *`address`* `-p` *`port`* `[--interface` *`interface`*`]`
- `IceDiscovery.Reply.Endpoints=udp [--interface` *`interface`*`]`

where

- *`address`* is the value of `IceDiscovery.Address` - defaults to `239.255.0.1` if IPv4 is enabled or `ff15::1` if IPv4 is disabled
- *`port`* is the value of `IceDiscovery.Port` - defaults to `4061`
- *`interface`* is the value of `IceDiscovery.Interface`

Consequently, if you don't define any of these properties, the plug-in uses the following endpoints by default (assuming IPv4):

- `IceDiscovery.Multicast.Endpoints=udp -h 239.255.0.1 -p 4061`
- `IceDiscovery.Reply.Endpoints=udp`

Finally, you can also override the default endpoint that a client uses to broadcast its lookup queries by defining `IceDiscovery.Lookup`, otherwise the plug-in computes this endpoint as follows:

- `IceDiscovery.Lookup=udp -h` *`address`* `-p` *`port`* `[--interface` *`interface`*`]`

This endpoint must use the same address and port as `IceDiscovery.Multicast.Endpoints`.

As you can see, the properties `IceDiscovery.Address`, `IceDiscovery.Port` and `IceDiscovery.Interface` are simply used as convenient shortcuts for customizing the details of the plug-in's endpoints. For example, suppose we want to use a different multicast address and port:

```
IceDiscovery.Address=239.255.0.99
IceDiscovery.Port=8000
```

The plug-in derives the following properties from these settings:

```
IceDiscovery.Multicast.Endpoints=udp -h 239.255.0.99 -p 8000
IceDiscovery.Lookup=udp -h 239.255.0.99 -p 8000
```

All of the clients and servers comprising an application must use the same values for `IceDiscovery.Address` and `IceDiscovery.Port`. You should also consider defining `IceDiscovery.DomainId` to avoid any potential collisions from unrelated applications that happen to use the same address and port.

Back to Top ^

Configuring IceDiscovery in Clients

Aside from installing the plug-in and optionally configuring its addressing information, no other configuration steps are required for an IceDiscovery client.

Back to Top ^

Configuring IceDiscovery in Servers

In addition to installing the plug-in and optionally configuring its addressing information, you also need to configure an identifier for each of a server's object adapters that hosts well-known (discoverable) objects. For example, suppose a server creates an object adapter named `Hello` and we want its objects to be discoverable. We can configure the object adapter's `AdapterId` property as follows:

```
Hello.AdapterId=HelloAdapter
```

The identifier you select must be globally unique among the servers sharing the same address and domain settings.

To use object adapter replication, you'll need to include the `ReplicaGroupId` property for each replicated object adapter:

```
Hello.AdapterId=Hello1
Hello.ReplicaGroupId=HelloAdapter
```

The indirect proxy `someObject@Hello1` refers to an object in this particular object adapter, whereas the indirect proxy `someObject@HelloAdapter` could refer to any object having the identity `someObject` in any of the object adapters participating in the replica group `HelloAdapter`.

Back to Top ^

Configuring a Locator Proxy

The IceDiscovery plug-in calls `setDefaultLocator` on its communicator at startup, therefore it's not necessary for you to configure a locator proxy.

Back to Top ^

Using IceDiscovery

As its name implies, the IceDiscovery plug-in enables clients to locate objects at run time, as long as the servers hosting those objects are actively running and using the same configuration settings for address, port, domain, and so on. Since IceDiscovery relies on UDP multicast to broadcast the lookup requests, you'll need to ensure that your network supports this transport.

IceDiscovery Design Decisions

From a design perspective, incorporating IceDiscovery into your application requires answering the following questions:

- What is the set of well-known objects that will be available for discovery?
  Applications typically don't need to make *every* object available for discovery. Rather, designs often only make "bootstrap" or "factory" objects available for discovery, while proxies for other objects can be obtained by invoking operations on these initial objects.

- How should clients refer to these well-known objects?
  As we explained earlier, proxies for well-known objects can take two forms: an identity by itself, or an identity with an adapter identifier, such as `factory` and `factory@AccountAdapter`, respectively. Clearly, using only identities places a greater burden on the application to ensure that they are globally unique among all of the clients and servers sharing the same address and domain settings. Including an object adapter identifier can help to further partition the object namespace, so that `factory@AccountAdapter` and `factory@AdminAdapter` represent distinct objects without requiring artificially unique identities such as `accountFactory` and `adminFactory`.

- Can unrelated applications share the same multicast address and port?
  If so, select unique domain identifiers for the applications and configure `IceDiscovery.DomainId` properties to avoid the potential for subtle bugs.

- Do you need replicated objects?
  Replicated object adapters can improve the fault tolerance of your application by allowing independent server processes to implement the same logical objects. Configure your servers as described above, and ensure your clients resolve indirect proxies using the replica group identifiers.

IceDiscovery provides enough flexibility to support a wide variety of application architectures. If you need additional functionality, consider using IceGrid instead. Note also that IceGrid supports its own version of IceDiscovery, so that migrating an existing IceDiscovery application should be straightforward.

Back to Top ^

IceDiscovery Sample Programs

The Ice distribution includes two IceDiscovery sample programs:

- `hello` - A basic client/server application
- `replication` - An application that demonstrates the use of object adapter replication

Refer to the `README` files in the demo source directories for more information on these examples.

Back to Top ^

See Also

- IceDiscovery.*
- Plug-in Facility
- Locators
- Well-Known Objects
- IceGrid