Object Adapter Endpoints

Previous Next

An object adapter maintains two sets of transport endpoints. One set identifies the network interfaces on which the adapter listens for new connections, and the other set is embedded in proxies created by the adapter and used by clients to communicate with it. We will refer to these sets of endpoints as the *physical endpoints* and the *published endpoints*, respectively. In most cases these sets are identical, but there are situations when they must be configured independently.

On this page:

Physical Object Adapter Endpoints

An object adapter's physical endpoints identify the network interfaces on which it receives requests from clients. These endpoints are configured via the *name*`.Endpoints` property, or they can be specified explicitly when creating an adapter using the operation `createObjectAdapterWithEndpoints`. The endpoint syntax generally consists of a transport protocol followed by an optional host name and port.

If a host name is specified, the object adapter listens only on the network interface associated with that host name. If no host name is specified but the property `Ice.Default.Host` is defined, the object adapter uses the property's value as the host name. Finally, if a host name is not specified, and the property `Ice.Default.Host` is undefined, the object adapter listens on all available network interfaces, including the loopback interface. You may also force the object adapter to listen on all interfaces by using one of the host names `0.0.0.0` or `*`. The adapter does *not* expand the list of interfaces when it is initialized. Instead, if no host is specified, or you use `-h *` or `-h 0.0.0.0`, the adapter binds to `INADDR_ANY` to listen for incoming requests.

If the host name refers to a DNS name which is configured with multiple addresses, the object adapter will listen on the network interfaces identified by each address. All the addresses should refer to local network interfaces or the object adapter creation will fail.

If you want an adapter to accept requests on certain network interfaces, you must specify a separate endpoint for each interface. For example, the following property configures a single endpoint for the adapter named `MyAdapter`:

```
MyAdapter.Endpoints=tcp -h 10.0.1.1 -p 9999
```

This endpoint causes the adapter to accept requests on the network interface associated with the IP address `10.0.1.1` at port `9999`. Note however that this adapter configuration does not accept requests on the loopback interface (the one associated with address `127.0.0.1`). If both addresses must be supported, then both must be specified explicitly, as shown below:

```
MyAdapter.Endpoints=tcp -h 10.0.1.1 -p 9999:tcp -h 127.0.0.1 -p 9999
```

If these are the only two network interfaces available on the host, then a simpler configuration omits the host name altogether, causing the object adapter to listen on both interfaces automatically:

```
MyAdapter.Endpoints=tcp -p 9999
```

If you want to make your configuration more explicit, you can use one of the special host names mentioned earlier:

```
MyAdapter.Endpoints=tcp -h * -p 9999
```

One advantage of this last example is that it ensures the object adapter *always* listens on all interfaces, even if a definition for `Ice.Default.Host` is later added to your configuration. Without an explicit host name, the addition of `Ice.Default.Host` could potentially change the interfaces on which the adapter is listening. For diagnostic purposes, you can determine the set of local addresses that Ice substitutes for the wildcard address by setting the property `Ice.Trace.Network=3` and reviewing the server's log output.

Careful consideration must also be given to the selection of a port for an endpoint. If no port is specified, the adapter uses a port that is selected (essentially at random) by the operating system, meaning the adapter will likely be using a different port each time the server is restarted. Whether that behavior is desirable depends on the application, but in many applications a client has a proxy containing the adapter's endpoint and expects that proxy to remain valid indefinitely. Therefore, an endpoint generally should contain a fixed port to ensure that the adapter is always listening at the same port.

However, there are certain situations where a fixed port is not required. For example, an adapter whose servants are transient does not need a fixed port, because the proxies for those objects are not expected to remain valid past the lifetime of the server process. Similarly, a server using indirect binding via IceGrid does not need a fixed port because its port is never published.

Back to Top ^

Published Object Adapter Endpoints

An object adapter publishes its endpoints in the proxies it creates, but it is not always appropriate to publish the adapter's physical endpoints in a proxy. For example, suppose a server is running on a host in a private network, protected from the public network by a firewall that can forward network traffic to the server. The adapter's physical endpoints must use the private network's address scheme, but a client in the public network would be unable to use those endpoints if they were published in a proxy. In this scenario, the adapter must publish endpoints in its proxies that direct the client to the firewall instead.

The published endpoints are configured using the adapter property `name.PublishedEndpoints`.

If this property is not defined, the adapter publishes its physical endpoints by default, with two exceptions:

- endpoints for the loopback address (`127.0.0.1`) are not published unless the loopback interface is the only interface, or `127.0.0.1` (or `loopback`) is explicitly listed as an endpoint with the `-h` option. Otherwise, to force the inclusion of loopback endpoints when they would normally be excluded, you must define *name*`.PublishedEndpoints` explicitly,
- endpoints with DNS names and no fixed port number are expanded to multiple endpoints if the DNS name refers to multiple addresses. There will be one published endpoint with the system allocated port per address. For example, `tcp -h corphost` could be expanded to `tcp -h 192.168.1.64 -p 64567:tcp -h 192.168.2.67 -p 64568`.

As an example, the properties below configure the adapter named `MyAdapter` with physical and published endpoints:

```
MyAdapter.Endpoints=tcp -h 10.0.1.1 -p 9999
MyAdapter.PublishedEndpoints=tcp -h corpfw -p 25000
```

This example assumes that clients connecting to host `corpfw` at port `25000` are forwarded to the adapter's endpoint in the private network.

Another use case of published endpoints is for replicated servers. Suppose we have two instances of a stateless server running on separate hosts in order to distribute the load between them. We can supply the client with a bootstrap proxy containing the endpoints of both servers, and the Ice run time in the client will select one of the servers at random when a connection is established. However, should the client invoke an operation on a server that returns a proxy for another object, that proxy would normally contain only the endpoint of the server that created it. Invocations on the new proxy would always be directed at the same server, reducing the opportunity for load balancing.

We can alleviate this situation by configuring the adapters to publish the endpoints of both servers. For example, here is a configuration for the server on host `Sun1`:

```
MyAdapter.Endpoints=tcp -h Sun1 -p 9999
MyAdapter.PublishedEndpoints=tcp -h Sun1 -p 9999:tcp -h Sun2 -p 9999
```

Similarly, the configuration for host `Sun2` retains the same published endpoints:

```
MyAdapter.Endpoints=tcp -h Sun2 -p 9999
MyAdapter.PublishedEndpoints=tcp -h Sun1 -p 9999:tcp -h Sun2 -p 9999
```

For troubleshooting purposes, you can examine the published endpoints for an object adapter by setting the property `Ice.Trace.Network=3`. Note however that this setting generates significant trace information about the Ice run time's network activity, therefore you may not want to use this setting by default.

You can also change the published endpoints at run time, by calling `setPublishedEndpoints` on `ObjectAdapter`:

Slice

```
local interface ObjectAdapter
{
    void setPublishedEndpoints(EndpointSeq newEndpoints);
    // ...
}
```

Back to Top ^

Refreshing Object Adapter Endpoints

A host's network interfaces may change over time, for example, if a laptop moves in and out of range of a wireless network. The object adapter provides an operation to refresh its list of interfaces:

Slice

```
local interface ObjectAdapter
{
    void refreshPublishedEndpoints();
    // ...
}
```

Calling `refreshPublishedEndpoints` causes the object adapter to update its internal list of available network interfaces and to use this updated information in the *`name`*`.PublishedEndpoints` property. This allows you to react to changing network interfaces while an object adapter is in use, but your application code is responsible for determining when it is necessary to call this operation.

Note that `refreshPublishedEndpoints` takes effect only for object adapters that specify published endpoints without a host, or that set the published endpoints to `-h *` or `-h 0.0.0.0`.

Back to Top ^

Timeouts in Object Adapter Endpoints

As a defense against hostile clients, we recommend that you specify a timeout for your physical object adapter endpoints. The timeout value you select affects tasks that the Ice run time normally does not expect to block for any significant amount of time, such as writing a reply message to a socket or waiting for SSL negotiation to complete. If you don't specify a timeout for an endpoint, the Ice run time uses the value of the `Ice.Default.Timeout` prop erty. You can also specify the value `infinite` in an endpoint to wait indefinitely, but be aware that this could allow malicious or misbehaving clients to consume excessive resources such as file descriptors.

Specifying a timeout in an object adapter endpoint is done exactly as in a proxy endpoint, using the `-t` option:

```
MyAdapter.Endpoints=tcp -p 9999 -t 5000
```

In this example, we specify a timeout of five seconds.

Unless overridden by published endpoints, the timeout specified in your object adapter endpoint also appears in the endpoints of any proxies created by that object adapter. This feature allows you to provide a default timeout value for clients that use your object adapter, although the client's code or configuration can cause it to use a different timeout in practice.

Back to Top ^

Discovering Object Adapter Endpoints

The object adapter provides operations for retrieving the physical and published endpoints:

Slice

```
module Ice
{
    local interface ObjectAdapter
    {
        // ...

        EndpointSeq getEndpoints();
        EndpointSeq getPublishedEndpoints();

        // ...
    }
}
```

The sequences that are returned contain `Endpoint` objects representing the adapter's physical and published endpoints, respectively.

Back to Top ^

A Router's Effect on Object Adapter Endpoints

If an object adapter is configured with a router, the adapter's published endpoints are augmented to reflect the router. See Glacier2 for more information on configuring an adapter with a router.

Back to Top ^

See Also

- Proxy and Endpoint Syntax
- Object Adapter Properties
- Ice.Default.*
- IceGrid
- Using Connections
- Communicators
- Glacier2

Previous Next