Proxy and Endpoint Syntax

On this page:

Syntax for Stringified Proxies

Synopsis

identity -f *facet* -e *encoding* -p *protocol* -t -o -O -d -D -s @ *adapter_id* : *endpoints*

Description

A stringified proxy consists of an identity, proxy options, and an optional object adapter identifier or endpoint list. White space (the space, tab (\t), line feed (\n), and carriage return (\r) characters) act as token delimiters; if a white space character appears as part of a component of a stringified proxy (such as the identity), it must be quoted or escaped as described below.

A proxy containing an identity with no endpoints is a well-known proxy; a proxy with an identity and an object adapter identifier represents an indirect proxy that will be resolved using the Ice locator.

Proxy options configure the invocation mode:

-f *facet*	Select a facet of the Ice object.
-e *encoding*	Specify the Ice encoding version to use when an invocation on this proxy marshals parameters.
-p *protocol*	Specify the Ice protocol version to use when sending a request with this proxy.
-t	Configures the proxy for twoway invocations (default).
-o	Configures the proxy for oneway invocations.
-O	Configures the proxy for batch oneway invocations.
-d	Configures the proxy for datagram invocations.
-D	Configures the proxy for batch datagram invocations.
-s	Configures the proxy for secure invocations.

The proxy options -t, -o, -O, -d, and -D are mutually exclusive.

The object identity identity is structured as [*category*/]*name*, where the *category* component and slash separator are optional. If *identity* contains white space or either of the characters : or @, it must be enclosed in single or double quotes. The *category* and *name* components are strings that are encoded as described in Object Identity, in particular, any occurrence of a slash (/) in *category* or *name* must be escaped with a backslash (i.e., \/).

The *facet* argument of the -f option represents a facet name. If *facet* contains white space, it must be enclosed in single or double quotes. A facet name is a string encoded like a Slice String Literal.

Likewise, an object adapter identifier *adapter_id* is a string encoded like a Slice String Literal. If *adapter_id* contains white space, it must be enclosed in single or double quotes.

Single or double quotes can be used to prevent white space characters from being interpreted as delimiters. Double quotes prevent interpretation of a single quote as an opening or closing quote, for example:

```
"a string with a ' quote"
```

Single quotes prevent interpretation of a double quote as an opening or closing quote. For example:

```
'a string with a " quote'
```

If *endpoints* are specified, they must be separated with a colon (`:`) and formatted as described in the endpoint syntax. The order of endpoints in the stringified proxy is not necessarily the order in which connections are attempted during binding: when a stringified proxy is converted into a proxy instance, by default, the endpoint list is randomized as a form of load balancing. You can change this default behavior using the properties `Ice.Default.EndpointSelection` and *name*`.EndpointSelection`.

If the `-s` option is specified, only those endpoints that support secure invocations are considered during binding. If no valid endpoints are found, the application receives `Ice::NoEndpointException`.

Otherwise, if the `-s` option is not specified, the endpoint list is ordered so that non-secure endpoints have priority over secure endpoints during binding. In other words, connections are attempted on all non-secure endpoints before any secure endpoints are attempted.

The `-e` and `-p` options specify the encoding and protocol versions supported by the target object, respectively. The Ice run time in the client is responsible for using a compatible version of the encoding or protocol, where *compatible* means the same major version and a minor version that is equal to or less than the specified minor version. If the `-e` option is not defined, the proxy uses the default version specified by `Ice.Default.EncodingVersion`.

If an unknown option is specified, or the stringified proxy is malformed, the application receives `Ice::ProxyParseException`. If an endpoint is malformed, the application receives `Ice::EndpointParseException`.

Syntax for Stringified Endpoints

Synopsis

endpoint `:` *endpoint*

Description

An endpoint list comprises one or more endpoints separated by a colon (`:`).

An endpoint has the following format:

transport option

`transport` can be any of the Ice transport protocols described on this page (`tcp`, `ssl`, `udp` etc.), or `default`. When `default` is used, it is replaced by the value of the `Ice.Default.Protocol` property. If an endpoint is malformed, or an unknown transport protocol is specified, the application receives `Ice::EndpointParseException`. The `ssl` transport protocol and transport protocols with a `s` suffix such as `wss` and `bts` are only available if the IceSSL plug-in is installed.

Ice uses endpoints for two similar but distinct purposes:

1. In a client context (that is, in a proxy), endpoints determine how Ice establishes a connection to a server.
2. In a server context (that is, in an object adapter's configuration), endpoints define the addresses and transport protocols over which new incoming connections are accepted. These endpoints are also embedded in the proxies created by the object adapter, unless a separate set of "published" endpoints are explicitly configured.

The sections that follow discuss the addressing component of endpoints, as well as the transport protocols and their supported options.

See Object Adapter Endpoints for examples.

IP Address Syntax

Synopsis

host `:` *hostname* `|` *x.x.x.x* `(IPv4)`
host `:` *hostname* `|` *":x:x:x:x:x:x:x"* `(IPv6)`

Description

Ice supports Internet Protocol (IP) versions 4 and 6 in all language mappings.

Support for these transport protocols is configured using the properties `Ice.IPv4` and `Ice.IPv6` (both enabled by default).

In the endpoint descriptions below, the *`host`* parameter represents either a host name that is resolved via the Domain Name System (DNS), an IPv4 address in dotted quad notation, or an IPv6 address in 128-bit hexadecimal format and enclosed in double quotes. Due to limitation of the DNS infrastructure, host and domain names are restricted to the ASCII character set.

The presence (or absence) of the *`host`* parameter has a significant influence on the behavior of the Ice run time. The table below describes these semantics:

Value	Client Semantics	Server Semantics
None	If *`host`* is not specified in a proxy, Ice uses the value of the `Ice.Default.Host` property. If that property is not defined, outgoing connections are only attempted over loopback interfaces.	If *`host`* is not specified in an object adapter endpoint, Ice uses the value of the `Ice.Default.Host` property. If that property is not defined, the adapter behaves as if the wildcard symbol * was specified (see below).
Host name	The host name is resolved via DNS. Outgoing connections are attempted to each address returned by the DNS query.	The host name is resolved via DNS, and the object adapter listens on the network interfaces corresponding to each address returned by the DNS query. The specified host name is embedded in proxies created by the adapter if the endpoint specifies a fixed port. Otherwise, if the port is selected by the operating system, proxies will embed an endpoint for each address and the system allocated port.
IPv4 address	An outgoing connection is attempted to the given address.	The object adapter listens on the network interface corresponding to the address. The specified address is embedded in proxies created by the adapter.
IPv6 address	An outgoing connection is attempted to the given address.	The object adapter listens on the network interface corresponding to the address. The specified address is embedded in proxies created by the adapter.
`0.0.0.0` (IPv4)	A "wildcard" IPv4 address that causes Ice to try all local interfaces when establishing an outgoing connection.	The adapter listens on all IPv4 network interfaces (including the loopback interface), that is, binds to INADDR_ANY for IPv4. Endpoints for all addresses except loopback are published in proxies (unless loopback is the only available interface, in which case only loopback is published).
`"::"` (IPv6)	A "wildcard" IPv6 address that causes Ice to try all local interfaces when establishing an outgoing connection.	Equivalent to * (see below).
* (IPv4, IPv6)	Not supported in proxies.	The adapter listens on all network interfaces (including the loopback interface), that is, binds to INADDR_ANY for the enabled transport protocols (IPv4 and/or IPv6). Endpoints for all addresses except loopback and IPv6 link-local are published in proxies (unless loopback is the only available interface, in which case only loopback is published).

There is one additional benefit in specifying a wildcard address for *`host`* (or not specifying it at all) in an object adapter's endpoint: if the list of network interfaces on a host may change while the application is running, using a wildcard address for *`host`* ensures that the object adapter automatically includes the updated interfaces. Note however that the list of published endpoints is not changed automatically; rather, the application must explicitly refresh the object adapter's endpoints. For diagnostic purposes, you can set the configuration property `Ice.Trace.Network=3` to cause Ice to log the current list of local addresses that it is substituting for the wildcard address.

When IPv4 and IPv6 are enabled, an object adapter endpoint that uses an IPv6 (or * wildcard) address can accept both IPv4 and IPv6 connections.

Java's default network stack always accepts both IPv4 and IPv6 connections regardless of the settings of `Ice.IPv6`. Thus, in Java, an object adapter endpoint that uses the IPv4 wildcard will accept both IPv4 and IPv6 connections. You can configure the Java run time to use only IPv4 by starting your application with the following JVM option:

Java

```
java -Djava.net.preferIPv4Stack=true ...
```

TCP Endpoint Syntax

Synopsis

```
tcp -h host -p port -t timeout -z --sourceAddress addr
```

Availability

The TCP transport protocol is a built-in transport protocol: it's always available.

Description

A `tcp` endpoint supports the following options:

Option	Description	Client Semantics	Server Semantics
`-h` *`host`*	Specifies the host name or IP address of the endpoint. If not specified, the value of `Ice.Default.Host` is used instead.	See IP Address Syntax.	See IP Address Syntax.
`-p` *`port`*	Specifies the port number of the endpoint.	Determines the port to which a connection attempt is made (required).	The port will be selected by the operating system if this option is not specified or *`port`* is zero.
`-t` *`timeout`*	Specifies the endpoint timeout in milliseconds.	The value for *`timeout`* must either be `infinite` to specify no timeout, or an integer greater than zero representing the timeout in milliseconds used by the client to open or close connections and to read or write data. If a timeout occurs, the application receives `Ice::TimeoutException`. If this option is not specified, it defaults to the value of `Ice.Default.Timeout`.	The value for *`timeout`* must either be `infinite` to specify no timeout, or an integer greater than zero representing the timeout in milliseconds used by the server to accept or close connections and to read or write data (see Timeouts in Object Adapter Endpoints and Connection Timeouts). *`timeout`* also controls the timeout that is published in proxies created by the object adapter. If this option is not specified, it defaults to the value of `Ice.Default.Timeout`.
`-z`	Specifies bzip2 compression.	Determines whether compressed requests are sent.	Determines whether compression is advertised in proxies created by the adapter.
`--sourceAddress` *`ADDR`*	Specifies the source address used by the connection.	The value for *`ADDR`* must be a numeric IPv4 or IPv6 address. If this option is not specified, it defaults to the value of `Ice.Default.SourceAddress`. This option allows to specify the source address set in the IP packet. It doesn't necessarily imply that the operating system will use the network interface matching this IP address to send out the IP packet. This feature is not supported on Universal Windows (UWP).	Not supported

UDP Endpoint Syntax

Synopsis

```
udp -h host -p port -z --ttl TTL --interface INTF --sourceAddress addr
```

Availability

The UDP transport protocol is a built-in transport protocol except for C++ and Objective-C static builds where it's only available to programs that explicitly register the UDP plug-in. See Using Plugins with Static Libraries.

Description

A `udp` endpoint supports either unicast or multicast delivery; the address resolved by *`host`* argument determines the delivery mode. To use multicast in IPv4, select an IP address in the range `233.0.0.0 to 239.255.255.255`. In IPv6, use an address that begins with `ff`, such as `ff01::1:1`.

A `udp` endpoint supports the following options:

Option	Description	Client Semantics	Server Semantics
`-h` *`host`*	Specifies the host name or IP address of the endpoint. If not specified, the value of `Ice.Default.Host` is used instead.	See IP Address Syntax.	See IP Address Syntax.

`-p` *`port`*	Specifies the port number of the endpoint.	Determines the port to which datagrams are sent (required).	The port will be selected by the operating system if this option is not specified or port is zero.
`-z`	Specifies bzip2 compression.	Determines whether compressed requests are sent.	Determines whether compression is advertised in proxies created by the adapter.
`--ttl` *`TTL`*	Specifies the time-to-live (also known as "hops") of multicast messages.	Determines whether multicast messages are forwarded beyond the local network. If not specified, or the value of *`TTL`* is `-1`, multicast messages are not forwarded. The maximum value is `255`.	N/A
`--interface` *`INTF`*	Specifies the network interface or group for multicast messages (see below).	Selects the network interface for outgoing multicast messages. If not specified, multicast messages are sent using the default interface.	Selects the network interface to use when joining the multicast group. If set to * or not specified, the group is joined on all the local network interfaces.
`--sourceAddress` *`ADDR`*	Binds outgoing socket connections to the network interface associated with *`ADDR`*.	The value for *`ADDR`* must be a numeric IPv4 or IPv6 address. If this option is not specified, it defaults to the value of `Ice.Default.SourceAddress`. This option allows to specify the source address set in the IP packet. It doesn't necessarily imply that the operating system will use the network interface matching this IP address to send out the IP packet. This feature is not supported on Universal Windows (UWP).	Not supported

Deprecated options

With the 1.0 encoding, UDP endpoints supported 2 additional options: the -e *major.minor* and -v *major.minor* options. These 2 options specified which encoding and protocol was supported by the endpoint. These two options are deprecated with the 1.1 encoding and are ignored (a deprecation warning will be emitted when parsed by the communicator `stringToProxy` method). The supported protocol and encoding is specified on the proxy with the 1.1 encoding.

Multicast Interfaces

When *`host`* denotes a multicast address, the `--interface` *`INTF`* option selects a particular network interface to be used for communication. The format of *`INTF`* depends on the language and IP version:

- C++ and .NET
  `INTF` can be an interface name, such as `eth0`, or an IP address. If using IPv6, it can also be an interface index. Interface names on Windows may contain spaces, such as `Local Area Connection`, therefore they must be enclosed in double quotes.
- Java
  `INTF` can be an interface name, such as `eth0`, or an IP address. On Windows, Java maps interface names to Unix-style nicknames.

SSL Endpoint Syntax

Synopsis

```
ssl -h host -p port -t timeout -z --sourceAddress addr
```

Availability

The SSL transport protocol is provided by a separate IceSSL plug-in.

Description

An `ssl` endpoint supports the same options as for `tcp` endpoints.

WS Endpoint Syntax

Synopsis

```
ws -r resource -h host -p port -t timeout -z --sourceAddress addr
```

Availability

The WS transport protocol is a built-in transport protocol except for C++ and Objective-C static builds where it's only available to programs that explicitly register the WS plug-in. See Using Plugins with Static Libraries.

Description

A `ws` (WebSocket) endpoint supports all `tcp` endpoint options in addition to the following:

Option	Description	Client Semantics	Server Semantics
`-r resource`	A URI specifying the resource associated with this endpoint. If not specified, the default value is `/`.	The value for *`resource`* is passed as the target for `GET` in the WebSocket upgrade request.	The web server configuration must direct the given *`resource`* to this endpoint.

WSS Endpoint Syntax

Synopsis

```
wss -r resource -h host -p port -t timeout -z --sourceAddress addr
```

Availability

The WSS transport protocol is a built-in transport protocol that is enabled when IceSSL is configured on the communicator. Like for the WS transport protocol, the WS plug-in needs to be registered explicitly for C++ and Objective-C static builds. See Using Plugins with Static Libraries.

Description

A `wss` (Secure WebSocket) endpoint supports all `ssl` endpoint options in addition to the following:

Option	Description	Client Semantics	Server Semantics
`-r resource`	A URI specifying the resource associated with this endpoint. If not specified, the default value is `/`.	The value for *`resource`* is passed as the target for `GET` in the WebSocket upgrade request.	The web server configuration must direct the given *`resource`* to this endpoint.

Bluetooth Endpoint Syntax

Synopsis

```
bt -a addr -u uuid -c channel -t timeout -z --name name
bts -a addr -u uuid -c channel -t timeout -z --name name
```

Availability

The Bluetooth transport protocol is provided by a separate IceBT plug-in.

Description

Support for Bluetooth endpoints is provided by the IceBT transport plug-in. The plug-in enables the `bt` transport protocol by default; if the IceSSL plug-in is also installed, IceBT enables the `bts` transport protocol as well. A Bluetooth endpoint supports the following options:

Option	Description	Client Semantics	Server Semantics

`-a` *`addr`*	Specifies the Bluetooth device address in the form *`xx:xx:xx:xx:xx:xx`*. The address must be enclosed in quotes because the colon (`:`) character is also a separator in the endpoint syntax. If not specified, the value of `Ice.Default.Host` is used instead.	An address is required for proxy endpoints. The plug-in will throw `EndpointParseException` if no address is specified via the `-a` option or `Ice.Default.Host`.	On Linux, the plug-in uses the address to select the Bluetooth adapter on which to listen for new connections. If no address is specified via the `-a` option or `Ice.Default.Host`, the plug-in uses the system's default adapter. On Android, the plug-in ignores this setting and always uses the system's default adapter. In addition to a Bluetooth device address, *`addr`* can also be `*` (e.g., `bt -a * ...`) to force the endpoint to use the system's default adapter without having to specify its address directly. The plug-in ignores the setting for `Ice.Default.Host` in this case.
`-u` *`uuid`*	Specifies the UUID of a service.	A UUID is required for proxy endpoints. The plug-in uses the UUID to search for a matching service at the specified device address.	A UUID is optional (but recommended) for object adapter endpoints. If this option is not specified, the plug-in generates a random UUID. The plug-in registers the service with its UUID in the local SDP registry so that clients can locate it.
`-c` *`channel`*	Specifies the RFCOMM channel number of the endpoint.	Not supported in proxy endpoints – the plug-in always uses the service UUID to connect to a server.	This setting is optional on Linux. The value for *`channel`* must be in the range 0-30. An available channel will be selected automatically if this setting is not specified or *`channel`* is zero. This setting is ignored on Android – the system always selects the channel.
`-t` *`timeout`*	Specifies the endpoint timeout in milliseconds.	The value for *`timeout`* must either be `infinite` to specify no timeout, or an integer greater than zero representing the timeout in milliseconds used by the client to open or close connections and to read or write data. If a timeout occurs, the application receives `Ice::TimeoutException`. If this option is not specified, it defaults to the value of `Ice.Default.Timeout`.	The value for *`timeout`* must either be `infinite` to specify no timeout, or an integer greater than zero representing the timeout in milliseconds used by the server to accept or close connections and to read or write data (see Timeouts in Object Adapter Endpoints and Connection Timeouts). *`timeout`* also controls the timeout that is published in proxies created by the object adapter. If this option is not specified, it defaults to the value of `Ice.Default.Timeout`.
`-z`	Specifies bzip2 compression.	Determines whether compressed requests are sent.	Determines whether compression is advertised in proxies created by the adapter.
`--name` *`name`*	Specifies the service name.	Ignored in proxy endpoints.	Associates a human-friendly name with the service's entry in the SDP registry. If not specified, the default name is `Ice Service`.

iAP Endpoint Syntax

Synopsis

```
iap -p protocol -n name -m manufacturer -o model number -t timeout -z
iaps -p protocol -n name -m manufacturer -o model number -t
```

Availability

The iAP transport protocol is provided by a separate IceIAP plug-in.

Description

Support for iAP endpoints is provided by the IceIAP transport plug-in. The plug-in enables the `iap` transport protocol by default; if the IceSSL plug-in is also installed, IceIAP enables the `iaps` transport protocol as well. The iAP transport protocol allows applications running on iOS to communicate with accessories connected to the iOS device either through Bluetooth or the lightning connector. The iAP transport protocol is a client-side transport protocol: there's no server-side support. An iAP endpoint supports the following options to allow selecting the accessory to connect to:

Option	Description	Client Semantics
`-p` *`protocol`*	Specifies the protocol implemented by the accessory.	The iAP transport protocol will only connect to accessories that advertise the specified iAP protocol. The default value for protocol is `com.zeroc.ice`.
`-n` *`name`*	Specifies the name of the accessory.	When specified, the iAP transport protocol will only connect to accessories that match the specified accessory name.

`-m` *`manufacturer`*	Specifies the manufacturer of the accessory.	When specified, the iAP transport protocol will only connect to accessories that match the specified accessory manufacturer.
`-o` *`model number`*	Specifies the model number of the accessory.	When specified, the iAP transport protocol will only connect to accessories that match the specified accessory model number.
`-t` *`timeout`*	Specifies the endpoint timeout in milliseconds.	The value for *`timeout`* must either be `infinite` to specify no timeout, or an integer greater than zero representing the timeout in milliseconds used by the client to open or close connections and to read or write data. If a timeout occurs, the application receives `Ice::TimeoutException`. If this option is not specified, it defaults to the value of `Ice.Default.Timeout`.
`-z`	Specifies bzip2 compression.	Determines whether compressed requests are sent.

Opaque Endpoint Syntax

Synopsis

`opaque -t` *`type`* `-e` *`encoding`* `-v` *`value`*

Description

Proxies can contain endpoints that are not universally understood by Ice processes. For example, a proxy can contain an SSL endpoint; if that proxy is marshaled to a receiver without the IceSSL plug-in, the SSL endpoint does not make sense to the receiver.

Ice preserves such unknown endpoints when they are received over the wire. For the preceding example, if the receiver remarshals the proxy and sends it back to an Ice process that does have the IceSSL plug-in, that process can invoke on the proxy using its SSL transport. This mechanism allows proxies containing endpoints for arbitrary transports to pass through processes that do not understand these endpoints without losing information.

If an Ice process stringifies a proxy containing an unknown endpoint, it writes the endpoint as an opaque endpoint. For example:

```
opaque -t 2 -e 1.0 -v CTEyNy4wLjAuMREnAAD/////AA==
```

This is how a process without the IceSSL plug-in stringifies an SSL endpoint. When a process with the IceSSL plug-in unstringifies this endpoint and converts it back into a string, it produces:

```
ssl -h 127.0.0.1 -p 10001
```

An `opaque` endpoint supports the following options:

Option	Description
`-t` *`type`*	Specifies the transport protocol for the endpoint. Transports are indicated by positive integers (1 for TCP, 2 for SSL, 3 for UDP etc,).
`-e` *`encoding`*	Specifies the encoding version used to encode the endpoint marshaled data.
`-v` *`value`*	Specifies the marshaled encoding of the endpoint in base-64 encoding.

Exactly one each of the `-t` and `-v` options must be present in an opaque endpoint. If `-e` is not specified, the default encoding used by the Ice runtime is assumed.

See Also

- Ice Protocol and Encoding
- Object Adapter Endpoints
- Connection Timeouts
- Ice.Default.*