

Proxy and Endpoint Syntax

On this page:

- [Syntax for Stringified Proxies](#)
- [Syntax for Stringified Endpoints](#)
 - [Address Syntax](#)
 - [TCP Endpoint Syntax](#)
 - [UDP Endpoint Syntax](#)
 - [SSL Endpoint Syntax](#)
 - [WS Endpoint Syntax](#)
 - [WSS Endpoint Syntax](#)
 - [Opaque Endpoint Syntax](#)

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:

<code>-f facet</code>	Select a facet of the Ice object.
<code>-e encoding</code>	Specify the Ice encoding version to use when an invocation on this proxy marshals parameters.
<code>-p protocol</code>	Specify the Ice protocol version to use when sending a request with this proxy.
<code>-t</code>	Configures the proxy for twoway invocations (default).
<code>-o</code>	Configures the proxy for oneway invocations.
<code>-O</code>	Configures the proxy for batch oneway invocations.
<code>-d</code>	Configures the proxy for datagram invocations.
<code>-D</code>	Configures the proxy for batch datagram invocations.
<code>-s</code>	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 UTF-8 strings that use the encoding described below. 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 UTF-8 string that uses the encoding described below.

The object adapter identifier `adapter_id` is a UTF-8 string that uses the encoding described below. 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'
```

Escape sequences such as `\b` are interpreted within single and double quotes.

UTF-8 strings are encoded using ASCII characters for the ordinal range 32--126 (inclusive). Characters outside this range must be encoded using escape sequences (`\b`, `\f`, `\n`, `\r`, `\t`) or octal notation (e.g., `\007`). Single and double quotes can be escaped using a backslash, as can the backslash itself.

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:

```
protocol option
```

The supported protocols are `tcp`, `udp`, `ssl`, `ws`, `wss`, and `default`. If `default` is used, it is replaced by the value of the [Ice.Default.Protocol](#) property. If an endpoint is malformed, or an unknown protocol is specified, the application receives `Ice::EndpointParseException`. The `ssl` and `wss` protocols 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 transports 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 protocols and their supported options.



See [Object Adapter Endpoints](#) for examples.

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 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 <i>host</i> is not specified in a proxy, Ice uses the value of the <code>Ice.Default.Host</code> property. If that property is not defined, outgoing connections are only attempted over loopback interfaces.	If <i>host</i> is not specified in an object adapter endpoint, Ice uses the value of the <code>Ice.Default.Host</code> property. If that property is not defined, the adapter behaves as if the wildcard symbol <code>*</code> 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 interface corresponding to the first address returned by the DNS query. The specified host name is embedded in proxies created by the adapter.
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 <code>INADDR_ANY</code> 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 <code>*</code> (see below).
* (IPv4, IPv6)	Not supported in proxies.	The adapter listens on all network interfaces (including the loopback interface), that is, binds to <code>INADDR_ANY</code> for the enabled 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
<code>java -Djava.net.preferIPv4Stack=true ...</code>

TCP Endpoint Syntax

Synopsis

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

Description

A `tcp` endpoint supports the following options:

Option	Description	Client Semantics	Server Semantics
<code>-h host</code>	Specifies the host name or IP address of the endpoint. If not specified, the value of <code>Ice.Default.Host</code> is used instead.	See Address Syntax .	See Address Syntax .
<code>-p port</code>	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 <code>port</code> is zero.
<code>-t timeout</code>	Specifies the endpoint timeout in milliseconds.	The value for <code>timeout</code> must either be infinite to specify no timeout, or an integer greater than zero representing the <code>timeout</code> in milliseconds used by the client to open or close connections and to read or write data. If a timeout occurs, the application receives <code>Ice::TimeoutException</code> . If this option is not specified, it defaults to the value of <code>Ice.Default.Timeout</code> .	The value for <code>timeout</code> 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). <code>timeout</code> 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 <code>Ice.Default.Timeout</code> .
<code>-z</code>	Specifies bzip2 compression.	Determines whether compressed requests are sent.	Determines whether compression is advertised in proxies created by the adapter.
<code>--sourceAddress addr</code>	Binds outgoing socket connections to the network interface associated with <code>addr</code> .	The value for <code>addr</code> must be a numeric IPv4 or IPv6 address. If this option is not specified, it defaults to the value of <code>Ice.Default.SourceAddress</code> . This feature is not supported on WinRT.	Not supported

UDP Endpoint Syntax

Synopsis

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

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
<code>-h host</code>	Specifies the host name or IP address of the endpoint. If not specified, the value of <code>Ice.Default.Host</code> is used instead.	See Address Syntax .	See Address Syntax .
<code>-p port</code>	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.
<code>-z</code>	Specifies bzip2 compression.	Determines whether compressed requests are sent.	Determines whether compression is advertised in proxies created by the adapter.
<code>--ttl TTL</code>	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 <code>TTL</code> is <code>-1</code> , multicast messages are not forwarded. The maximum value is 255.	N/A

<code>--interface <i>INTF</i></code>	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 not specified, the group is joined on the default network interface.
<code>--sourceAddress <i>addr</i></code>	Binds outgoing socket connections to the network interface associated with a <i>addr</i> .	The value for <i>addr</i> must be a numeric IPv4 or IPv6 address. If this option is not specified, it defaults to the value of <code>Ice.Default.SourceAddress</code> . This feature is not supported on WinRT.	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
```

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
```

Description

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

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

WSS Endpoint Syntax

Synopsis

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

Description

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

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

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
<code>-t type</code>	Specifies the transport for the endpoint. Transports are indicated by positive integers (1 for TCP, 2 for SSL, and 3 for UDP).
<code>-e encoding</code>	Specifies the encoding version used to encode the endpoint marshaled data.
<code>-v value</code>	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 current encoding supported by the Ice runtime is assumed.

See Also

- [The Ice Protocol](#)
- [Object Adapter Endpoints](#)
- [Connection Timeouts](#)
- [Ice.Default.*](#)