IceStorm Quality of Service

An IceStorm subscriber specifies Quality of Service (QoS) parameters at the time of subscription. The supported QoS parameters are described in the sections below

On this page:

Reliability QoS for IceStorm

The QoS parameter `reliability` affects message delivery. The only legal values at this point are `ordered` and the empty string. If not specified, the default value is the empty string (meaning not ordered).

The `ordered` reliability QoS requires a twoway subscriber proxy. If you specify this reliability QoS, IceStorm will forward events in the order they are received but doesn't forward them immediately. Instead, IceStorm waits for the reply from the forwarding of an event before forwarding the next event. This guarantees that the subscriber will process the events in the same order as they were received even if its threading model doesn't serialize incoming requests.

Retry Count QoS for IceStorm

IceStorm automatically removes a subscriber if `ObjectNotExistException` or `NotRegisteredException` is raised while attempting to deliver an event. IceStorm considers these exceptions as indicators of a hard failure, after which it is unnecessary to continue event delivery.

For other kinds of failures, IceStorm uses the QoS parameter `retryCount` to determine when to remove a subscriber. A value of `-1` means IceStorm retries forever and never automatically removes a subscriber unless a hard failure occurs. A value of zero means IceStorm never retries and immediately removes the subscriber. For positive values, IceStorm decrements the subscriber's retry count for each failure and removes the subscriber once it reaches zero. Linked topics always have a configured retry count of `-1`. The default value of the `retryCount` parameter is zero.

A retry count of `-1` adds some resiliency to your IceStorm application by ignoring intermittent network failures such as `ConnectionRefusedException`. However, there is also some risk inherent in using a retry count of `-1` because an improperly configured subscriber may never be removed. For example, consider what happens when a subscriber registers using a transient endpoint: if that subscriber happens to terminate and resubscribe with a different endpoint, IceStorm will continue trying to deliver events to the subscriber at its old endpoint. IceStorm can only remove the subscriber if it receives a hard error, and that is only possible when the subscriber is reachable.

To use a retry count of `-1` successfully, the subscriber should either register with a fixed endpoint, or use IceGrid to take advantage of indirect proxies and automatic activation. Furthermore, if the subscriber is expected to function correctly after a restart of its process, the subscriber must use the same identity. The application can rely on the `subscribeAndGetPublisher` operation to raise `AlreadySubscribed` when the subscriber is already subscribed.

IceStorm QoS Example

The Slice type `IceStorm::QoS` is defined as a `dictionary` whose key and value types are both `string`, therefore the QoS parameter name and value are both represented as strings. The code we presented in our earlier subscriber example used an empty dictionary for the QoS argument, meaning default values are used. The C++ and Java examples shown below illustrate how to set the `reliability` parameter to `ordered`.

Here is the C++ example:

C++

```
IceStorm::QoS qos;
qos["reliability"] = "ordered";
topic->subscribeAndGetPublisher(qos, proxy->ice_twoway());
```

Here is the Java example:

Java

```
java.util.Map qos = new java.util.HashMap();
qos.put("reliability", "ordered");
topic.subscribeAndGetPublisher(qos, proxy.ice_twoway());
```

See Also

- IceGrid
- Implementing an IceStorm Subscriber
- IceStorm Delivery Modes
- Object Identity