The Metrics Facet

Previous Next

The `Metrics` facet implements the Instrumentation Facility to provide convenient access to metrics for the Ice run time and select Ice services. The metrics provided by this facet include the number of threads currently running and their state, the number of connections, information on invocations and dispatch, as well as connection establishment and endpoint name resolution.

On this page:

- Metrics Terminology
- Metrics Types
- The MetricsAdmin Interface
- Obtaining the Local Metrics Facet
- Metrics Attributes

Metrics Terminology

- metric: an "analytical measurement intended to quantify the state of a system", recorded by the Ice run time, such as bytes sent over a connection.
- metric name: the name of the metric, such as "number of connections".
- metrics map: a collection of metrics objects.
- metrics view: a collection of metrics maps. A view contains metrics maps of different types (e.g., connections and threads). Several metrics views can be configured with different purposes. For example, you can have a "debug" metrics view to get detailed metrics of each of the instrumented objects in the Ice communicator. This view can be enabled from time to time for debugging purposes but it's disabled most of the time. You could also have a more coarse-grained metrics view to collect data at a higher level, such as the amount of bytes received and sent by all the connections from the communicator. This metrics view can be enabled all the time.

Back to Top ^

Metrics Types

Metrics are specified as Slice classes defined in the `Ice/Metrics.ice` Slice file. All the metrics types are defined in the `IceMX` module.

The base class is `IceMX::Metrics`:

Slice

```
class Metrics
{
    string id;
    long total = 0;
    int current = 0;
    long totalLifetime = 0;
    int failures = 0;
}
```

A metrics object is an instance of `IceMX::Metrics` and represents metrics of one or more instrumented objects. An instrumented object can be anything that supports instrumentation. The Ice run time supports instrumentation of the following objects and activities:

- Threads
- Connections
- Invocations
- Dispatch
- Connection establishment
- Endpoint resolution

The `id` of a metric identifies the instrumented object(s). The `total` and `current` members specify the total and current number of instrumented objects created since the creation of the Ice communicator, respectively. The `totalLifetime` member is the sum of the lifetime of each instrumented object and `failures` is the number of failures that have occurred for the metrics object(s).

Failures are specified using a separate `IceMX::MetricsFailures` structure:

Slice

```
struct MetricsFailures
{
    string id;
    StringIntDict failures;
}
```

The `failures` dictionary provides the count of each type of failure for a metric identified by `id`. Failures are dependent on the instrumented objects. For example, failures for Ice connections are represented with the name of the exception that caused the connection to fail (e.g., `Ice::ConnectionLostException` or `Ice::TimeoutException`).

A metrics map is simply defined as a sequence of `IceMX::Metrics` objects, and a metrics view is defined as a dictionary of metrics map:

Slice

```
sequence<Metrics> MetricsMap;
dictionary<string, MetricsMap> MetricsView;
```

The key for the metrics view dictionary is a string that identifies the metrics map. The Ice run time supports the following metrics maps:

Metrics map name	Slice class	Description
Connection	`IceMX::ConnectionMetrics`	Connection metrics.
Thread	`IceMX::ThreadMetrics`	Thread metrics. The Ice run time instruments threads for the communicator's thread pools as well as threads used internally.
Invocation	`IceMX::InvocationMetrics`	Client-side proxy invocation metrics.
Dispatch	`IceMX::DispatchMetrics`	Server-side dispatch metrics.
EndpointLookup	`IceMX::Metrics`	Endpoint lookup metrics. For tcp, ssl and udp endpoints, this corresponds to the DNS lookups made to resolve the host names in endpoints.
ConnectionEstablishment	`IceMX::Metrics`	Connection establishment metrics.

A metrics map can also contain sub-metrics maps. An example is the `Invocation` metrics map, which provides a `Remote` sub-metrics map to record metrics associated with remote invocations. The Slice class for remote invocation metrics is `IceMX::Metrics`.

Back to Top ^

The `MetricsAdmin` Interface

The Slice interface `IceMX::MetricsAdmin` allows you to retrieve the metrics associated with the Ice communicator:

Slice

```
module IceMX
{
    exception UnknownMetricsView {}

    interface MetricsAdmin
    {
        Ice::StringSeq getMetricsViewNames(out Ice::StringSeq disableViews);
        void enableMetricsView(string name)
            throws UnknownMetricsView;
        void disableMetricsView(string name)
            throws UnknownMetricsView;
        MetricsView getMetricsView(string view, out long timestamp)
            throws UnknownMetricsView;
        MetricsFailuresSeq getMapMetricsFailures(string view, string map)
            throws UnknownMetricsView;
        MetricsFailures getMetricsFailures(string view, string map, string id)
            throws UnknownMetricsView;
    }
}
```

The `getMetricsViewName` operation retrieves the names of the configured enabled and disabled views. The enableMetricsView and disableMetricsView allow to enable and disable a specific view. Calling those methods is equivalent to setting the view Disabled property to 1 or 0.The `getMetricsView` operation returns the metrics for the given view. The `getMapMetricsFailures` and `getMetricsFailures` operations retrieve the metrics failures for a given map or metrics id.

Back to Top ^

Obtaining the Local Metrics Facet

We already showed how to obtain a proxy for a remote administrative facet, but suppose you want to interact with the facet in your local address space. The code below shows the necessary steps:

C++11

```
auto obj = communicator->findAdminFacet("Metrics");
if(obj)
{
    // May be null if the facet is not enabled
    auto facet = std::dynamic_pointer_cast<Ice::MetricsAdmin>(obj);
    ...
}
```

C++98

```
Ice::ObjectPtr obj = communicator->findAdminFacet("Metrics");
if(obj)
{
    // May be null if the facet is not enabled
    Ice::MetricsAdminPtr facet = Ice::MetricsAdminPtr::dynamicCast(obj);
    ...
}
```

As shown here, the facet is registered with the name `Metrics` and a regular language cast is used to downcast the base object type to the `MetricsAdmin` interface.

Back to Top ^

Metrics Attributes

Metrics views are configured with IceMX Metrics properties.

The `GroupBy`, `Accept` and `Reject` properties are specified using attributes that are specific to each metrics map. The table below describes the attributes supported by the Ice run time's metrics maps.

Name	Maps	Description
id	All	A unique identifier to select the instrumented object or operation.
parent	All	The parent of the instrumented object or operation.
none	All	The none attribute is a special attribute that evaluates to the empty string.
endpoint	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	The stringified endpoint.
endpointType	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	The endpoint numerical type as defined in `Ice/Endpoint.ice`.
endpointIsDatagram	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	A boolean indicating if the endpoint is a datagram endpoint.
endpointIsSecure	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	A boolean indicating if the endpoint is secure.
endpointTimeout	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	The endpoint timeout.
endpointCompress	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	A boolean indicating if the endpoint requires compression.
endpointHost	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	The endpoint host.
endpointPort	Connection, Dispatch, Remote, ConnectionEstablishment, EndpointLookup	The endpoint port.
connection	Dispatch	The connection description.
incoming	Connection, Dispatch, Remote	A boolean where true indicates an incoming (server) connection and false an outgoing (client) connection.
adapterName	Connection, Dispatch, Remote	If the connection is a server connection, adapterName returns the name of the adapter that created the connection, otherwise it is the empty string.
connectionId	Connection, Dispatch, Remote	The ID of the connection if one is set, otherwise it is the empty string.
localAddress	Connection, Dispatch, Remote	The connection's local address.
localPort	Connection, Dispatch, Remote	The connection's local port.
remoteAddress	Connection, Dispatch, Remote	The connection's remote address.
remotePort	Connection, Dispatch, Remote	The connection's remote port.
mcastAddress	Connection, Dispatch, Remote	The connection's multicast address.
mcastPort	Connection, Dispatch, Remote	The connection's multicast port.
state	Connection	The state of the connection.
operation	Dispatch, Invocation	The dispatched or invoked operation name.
identity	Dispatch, Invocation	The identity of the Ice object used for the dispatch or invocation.
facet	Dispatch, Invocation	The facet of the Ice object used for the dispatch or invocation.
mode	Dispatch, Invocation	The dispatch or invocation mode.
context.*key*	Dispatch, Invocation	The value of the dispatch or invocation context with the given key.
proxy	Invocation	The proxy used for the invocation.
encoding	Invocation	The proxy encoding.

The `id`, `parent` and `none` attributes are supported by all maps.

The value of the `parent` attribute depends on the map. For the Connection, Dispatch and Remote maps, the `parent` will either be "Communicator" if the connection is a client connection, or the object adapter name if it's a server connection. For the Invocation, EndpointLookup, and ConnectionEstablishment maps, it will always be "Communicator". The `parent` attribute enables the filtering of metrics based on the object adapter. When used with the `GroupBy` property it also allows you to obtain metrics at the object adapter level. For instance, the following configuration does not monitor any metrics for the `Ice.Admin` object adapter and it groups all the metrics based on the object adapter or communicator:

```
IceMX.Metrics.MyView.GroupBy=parent
IceMX.Metrics.MyView.Reject.parent=Ice\.Admin   # Escape the dot in Ice.Admin
```

This configuration enables the communicator to get metrics on a per object adapter or communicator basis.

You can also use the `none` attribute to get metrics for the communicator including the metrics from object adapters, e.g., `IceMX.Metrics.MyView.GroupBy=none`. This provides the lowest possible level of detail as all the statistics will be recorded by a single metrics object.

The `id` attribute allows you to get a higher level of detail by recording metrics on a per instrumented object or operation basis. If you specify `IceMX.Metrics.MyView.GroupBy=id`, the `Metrics` facet will record metrics for each individual object or operation.

Back to Top ^

See Also

- Instrumentation Facility
- Creating the admin Object
- Ice.Admin.*

Previous

Next