Implementing an IceStorm Subscriber

Our weather measurement subscriber implementation takes the following steps:

1. Obtain a proxy for the `TopicManager`. This is the primary IceStorm object, used by both publishers and subscribers.
2. Create an object adapter to host our `Monitor` servant.
3. Instantiate the `Monitor` servant and activate it with the object adapter.
4. Subscribe to the `Weather` topic.
5. Process `report` messages until shutdown.
6. Unsubscribe from the `Weather` topic.

We present monitor implementations in C++ and Java below.

On this page:

Subscriber Example in C++

Our C++ monitor implementation begins by including the necessary header files. The interesting ones are `IceStorm/IceStorm.h`, which is generated from the IceStorm Slice definitions, and `Monitor.h`, containing the generated code for our monitor definitions:

C++

```
#include <Ice/Ice.h>
#include <IceStorm/IceStorm.h>
#include <Monitor.h>

using namespace std;

class MonitorI : virtual public Monitor {
public:
    virtual void report(const Measurement& m, const Ice::Current&) {
        cout << "Measurement report:" << endl
             << "  Tower: " << m.tower << endl
             << "  W Spd: " << m.windSpeed << endl
             << "  W Dir: " << m.windDirection << endl
             << "   Temp: " << m.temperature << endl
             << endl;
    }
};

int main(int argc, char* argv[])
{
    ...
    Ice::ObjectPrx obj = communicator->stringToProxy("IceStorm/TopicManager:tcp -p 9999");
    IceStorm::TopicManagerPrx topicManager = IceStorm::TopicManagerPrx::checkedCast(obj);

    Ice::ObjectAdapterPtr adapter = communicator->createObjectAdapter("MonitorAdapter");

    MonitorPtr monitor = new MonitorI;
    Ice::ObjectPrx proxy = adapter->addWithUUID(monitor)->ice_oneway();
    adapter->activate();

    IceStorm::TopicPrx topic;
    try {
        topic = topicManager->retrieve("Weather");
        IceStorm::QoS qos;
        topic->subscribeAndGetPublisher(qos, proxy);
    }
    catch (const IceStorm::NoSuchTopic&) {
        // Error! No topic found!
        ...
    }

    communicator->waitForShutdown();

    topic->unsubscribe(proxy);
    ...
}
```

Our implementation of the `Monitor` servant is currently quite simple. A real implementation might update a graphical display, or incorporate the measurements into an ongoing calculation.

C++

```
class MonitorI : virtual public Monitor {
public:
    virtual void report(const Measurement& m, const Ice::Current&) {
        cout << "Measurement report:" << endl
             << "  Tower: " << m.tower << endl
             << "  W Spd: " << m.windSpeed << endl
             << "  W Dir: " << m.windDirection << endl
             << "   Temp: " << m.temperature << endl
             << endl;
    }
};
```

After obtaining a proxy for the topic manager, the program creates an object adapter, instantiates the `Monitor` servant and activates it:

C++

```
    Ice::ObjectAdapterPtr adapter = communicator->createObjectAdapter("MonitorAdapter");

    MonitorPtr monitor = new MonitorI;
    Ice::ObjectPrx proxy = adapter->addWithUUID(monitor)->ice_oneway();
    adapter->activate();
```

Note that the code creates a oneway proxy for the `Monitor` servant. This is for efficiency reasons: by subscribing with a oneway proxy, IceStorm will deliver events to the subscriber via oneway messages, instead of via twoway messages. We also activate the object adapter at this time, which means the servant can now begin receiving invocations.

Next, the monitor subscribes to the topic:

C++

```
    IceStorm::TopicPrx topic;
    try {
        topic = topicManager->retrieve("Weather");
        IceStorm::QoS qos;
        topic->subscribeAndGetPublisher(qos, proxy);
    }
    catch (const IceStorm::NoSuchTopic&) {
        // Error! No topic found!
        ...
    }
```

Finally, the monitor blocks until the communicator is shutdown. After `waitForShutdown` returns, the monitor cleans up by unsubscribing from the topic:

C++

```
    adapter->activate();
    communicator->waitForShutdown();

    topic->unsubscribe(proxy);
```

Subscriber Example in Java

The Java implementation of the monitor is shown below:

Java

```
class MonitorI extends _MonitorDisp {
    public void report(Measurement m, Ice.Current curr) {
        System.out.println(
            "Measurement report:\n" +
            "  Tower: " + m.tower + "\n" +
            "  W Spd: " + m.windSpeed + "\n" +
            "  W Dir: " + m.windDirection + "\n" +
            "   Temp: " + m.temperature + "\n");
    }
}

public static void main(String[] args)
{
    ...
    Ice.ObjectPrx obj = communicator.stringToProxy("IceStorm/TopicManager:tcp -p 9999");
    IceStorm.TopicManagerPrx topicManager = IceStorm.TopicManagerPrxHelper.checkedCast(obj);

    Ice.ObjectAdapterPtr adapter = communicator.createObjectAdapter("MonitorAdapter");

    Monitor monitor = new MonitorI();
    Ice.ObjectPrx proxy = adapter.addWithUUID(monitor).ice_oneway();
    adapter.activate();

    IceStorm.TopicPrx topic = null;
    try {
        topic = topicManager.retrieve("Weather");
        java.util.Map qos = null;
        topic.subscribeAndGetPublisher(qos, proxy);
    }
    catch (IceStorm.NoSuchTopic ex) {
        // Error! No topic found!
        ...
    }

    communicator.waitForShutdown();

    topic.unsubscribe(proxy);
    ...
}
```

Our implementation of the `Monitor` servant is currently quite simple. A real implementation might update a graphical display, or incorporate the measurements into an ongoing calculation.

Java

```
class MonitorI extends _MonitorDisp {
    public void report(Measurement m, Ice.Current curr) {
        System.out.println(
            "Measurement report:\n" +
            "  Tower: " + m.tower + "\n" +
            "  W Spd: " + m.windSpeed + "\n" +
            "  W Dir: " + m.windDirection + "\n" +
            "   Temp: " + m.temperature + "\n");
    }
}
```

After obtaining a proxy for the topic manager, the program creates an object adapter, instantiates the `Monitor` servant and activates it:

Java

```
Monitor monitor = new MonitorI();
Ice.ObjectPrx proxy = adapter.addWithUUID(monitor).ice_oneway();
adapter.activate();
```

Note that the code creates a oneway proxy for the `Monitor` servant. This is for efficiency reasons: by subscribing with a oneway proxy, IceStorm will deliver events to the subscriber via oneway messages, instead of via twoway messages. We also activate the object adapter at this time, which means the servant can now begin receiving invocations.

Next, the monitor subscribes to the topic:

Java

```
IceStorm.TopicPrx topic = null;
try {
    topic = topicManager.retrieve("Weather");
    java.util.Map qos = null;
    topic.subscribeAndGetPublisher(qos, proxy);
}
catch (IceStorm.NoSuchTopic ex) {
    // Error! No topic found!
    ...
}
```

Finally, the monitor blocks until the communicator is shutdown. After `waitForShutdown` returns, the monitor cleans up by unsubscribing from the topic:

Java

```
communicator.waitForShutdown();

topic.unsubscribe(proxy);
```

See Also

- Using IceStorm
- Oneway Invocations