Stats Facility

The Ice run time uses the `Ice::Stats` interface to report the number of bytes sent and received over the wire on every operation invocation:

Slice

```
module Ice {
    local interface Stats {
        void bytesSent(string protocol, int num);
        void bytesReceived(string protocol, int num);
    };

    local interface Communicator {
        Stats getStats();
        // ...
    };
};
```

The Ice run time calls `bytesReceived` as it reads data from the network and `bytesSent` as it writes data to the network. A very simple implementation of the `Stats` interface could look as follows:

C++

```
class MyStats : public virtual Ice::Stats {
public:
    virtual void bytesSent(const string& prot, Ice::Int num)
    {
        cerr << prot << ": sent " << num << " bytes" << endl;
    }

    virtual void bytesReceived(const string& prot, Ice::Int)
    {
        cerr << prot << ": received " << num << " bytes" << endl;
    }
};
```

To register your implementation, you must pass it in an `InitializationData` parameter when you initialize a communicator:

C++

```
Ice::InitializationData id;
id.stats = new MyStats;
Ice::CommunicatorPtr ic = Ice::initialize(id);
```

You can install a `Stats` object on either the client or the server side (or both). Here is some example output produced by installing a `MyStats` object in a simple server:

```
tcp: received 14 bytes
tcp: received 32 bytes
tcp: sent 26 bytes
tcp: received 14 bytes
tcp: received 33 bytes
tcp: sent 25 bytes
...
```

In practice, your `Stats` implementation will probably be a bit more sophisticated: for example, the object can accumulate statistics in member variables and make the accumulated statistics available via member functions, instead of simply printing everything to the standard error output.

See Also