Parameter Passing in C++

For each parameter of a Slice operation, the C++ mapping generates a corresponding parameter for the virtual member function in the skeleton. In addition, every operation has a trailing parameter of type `Ice::Current`. For example, the `name` operation of the `Node` interface has no parameters, but the `name` member function of the `Node` skeleton class has a single parameter of type `Ice::Current`. We will ignore this parameter for now.

Parameter passing on the server side follows the rules for the client side:

- in-parameters are passed by value or `const` reference
- out-parameters are passed by reference
- return values are passed by value
- optional parameters are enclosed in `IceUtil::Optional` values

To illustrate the rules, consider the following interface that passes string parameters in all possible directions:

Slice

```
module M {
    interface Example {
        string op(string sin, out string sout);
    };
};
```

The generated skeleton class for this interface looks as follows:

C++

```
namespace M {
    class Example : virtual public ::Ice::Object {
    public:
        virtual std::string op(const std::string&, std::string&,
                               const Ice::Current& = Ice::Current()) = 0;
        // ...
    };
}
```

As you can see, there are no surprises here. For example, we could implement `op` as follows:

C++

```
std::string
ExampleI::op(const std::string& sin, std::string& sout, const Ice::Current&)
{
    cout << sin << endl;        // In parameters are initialized
    sout = "Hello World!";      // Assign out parameter
    return "Done";              // Return a string
}
```

This code is in no way different from what you would normally write if you were to pass strings to and from a function; the fact that remote procedure calls are involved does not impact on your code in any way. The same is true for parameters of other types, such as proxies, classes, or dictionaries: the parameter passing conventions follow normal C++ rules and do not require special-purpose API calls or memory management.

See Also

- Server-Side C++ Mapping for Interfaces
- C++ Mapping for Operations
- C++ Mapping for Optional Values
- Raising Exceptions in C++
- The Current Object