C++98 Mapping for Optional Values

Previous Next

On this page:

The `IceUtil::Optional` Template

The C++ mapping uses a template to hold the values of optional data members and parameters:

C++

```
namespace IceUtil
{
    template<typename T>
    class Optional
    {
    public:
        typedef T element_type;

        Optional();
        Optional(NoneType);
        Optional(const T&);

        template<typename Y>
        Optional(const Optional<Y>&);

        Optional(const Optional& r);

        Optional& operator=(NoneType);
        Optional& operator=(const T&);

        template<typename Y>
        Optional& operator=(const Optional<Y>&);
        Optional& operator=(const Optional&);

        const T& get() const;
        T& get();
        const T* operator->() const;
        T* operator->();
        const T& operator*() const;
        T& operator*();

        operator bool() const;
        bool operator!() const;

        void swap(Optional& other);

        ...
    };
}
```

The `IceUtil::Optional` template provides constructors and assignment operators that allow you to initialize an instance using the element type or an existing optional value. The default constructor initializes an instance to an unset condition. The `get` method and dereference operators retrieve the current value held by the instance, or throw `IceUtil::OptionalNotSetException` if no value is currently set. Use the `bool` or `!` operators to test whether the instance has a value prior to dereferencing it. Finally, the `swap` method exchanges the state of two instances.

The `IceUtil::None` Value

The template includes a constructor and assignment operator that accept `NoneType`. Ice defines an instance of this type, `IceUtil::None`, that you can use to initialize (or reset) an `Optional` instance to an unset condition:

C++

```
IceUtil::Optional<int> i = 5;
i = IceUtil::None;
assert(!i); // true
```

You can pass `IceUtil::None` anywhere an `IceUtil::Optional` value is expected.

Previous Next

See Also

- Optional Data Members
- Operations
- C++98 Mapping for Operations