C-Sharp Mapping for Optional Values

On this page:

The `Ice.Optional` Type

The C# mapping uses a generic structure to hold the values of optional data members and parameters:

C#

```
namespace Ice
{
    public struct NoneType {}

    public struct Optional<T> {
        public Optional(NoneType none);
        public Optional(T v);
        public Optional(Optional<T> v);

        public static explicit operator T(Optional<T> v);
        public static implicit operator Optional<T>(T v);
        public static implicit operator Optional<T>(NoneType v);

        public T Value { get; }
        public bool HasValue { get; }

        public override bool Equals(object other);
        public override int GetHashCode();

        ...
    }
}
```

The `Ice.Optional` type provides constructors and conversion 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 `Value` property and conversion operator retrieve the current value held by the instance, or throw `System.InvalidOperationException` if no value is currently set. Use the `HasValue` property to test whether the instance has a value prior to accessing it.

The implicit conversion operators allow you to initialize an `Optional` instance without a cast. However, the conversion operator that extracts the current value is declared as `explicit` and therefore requires a cast:

C#

```
Ice.Optional<int> i = 5; // No cast necessary
int j = i;               // Error!
int k = (int)i;          // OK
```

This operator raises an exception if no value is set, so the proper way to write this is shown below:

C#

```
Ice.Optional<int> i = 5;
int j;
if(i.HasValue)
    j = (int)i;
else
    ...
```

The `Ice.Util.None` Value

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

C#

```
Ice.Optional<int> i = 5;
i = Ice.Util.None;
Debug.Assert(!i.HasValue); // true
```

You can pass `Ice.Util.None` anywhere an `Ice.Optional` value is expected.

See Also

- Optional Data Members