Optional Data Members

On this page:

Overview of Optional Data Members

As of Ice 3.5, a data member of a Slice class or exception may be declared as optional to indicate that a program can leave its value unset. Data members not declared as optional are known as *required* members; a program must supply legal values for all required members.

Declaring Optional Data Members

Each optional data member in a type must be assigned a unique, non-negative integer *tag*:

Slice

```
class C
{
    string name;
    bool active;
    optional(2) string alternateName;
    optional(5) int overrideCode;
};
```

It is legal for a base type's tag to be reused by a derived type:

Slice

```
exception Base
{
    optional(1) int systemCode;
};

exception Derived extends Base
{
    optional(1) string diagnostic; // OK
};
```

The scope of a tag is limited to its enclosing type and has no effect on base or derived types.

Language mappings specify an API for setting an optional member and testing whether a member is set. Here is an example in C++:

C++

```
CPtr c = new C;
c->name = "xyz";           // required
c->active = true;          // required
c->alternateName = "abc";  // optional
c->overrideCode = 42;      // optional

if(c->alternateName)
    cout << "alt name = " << c->alternateName << endl;
```

As you can see, the C++ language mapping makes setting an optional member as simple as assigning it a value. Refer to the language mapping sections for more details on the optional data member API.

A well-behaved program must test for the presence of an optional member and not assume that it is always set. Dereferencing an unset optional member causes a run-time error.

In all supported language mappings, an optional data member's initial condition is unset if not otherwise assigned during construction. Again using C++ as an example:

C++

```
CPtr c = new C;            // default constructor
assert(!c->alternateName); // not set

c = new C("xyz", true, "abc", 42); // one-shot constructor
assert(c->alternateName);          // set by constructor
```

Optional Data Members with Default Values

You can declare a default value for optional members just as you can for required members:

Slice

```
class C
{
    string name;
    bool active = true;
    optional(2) string alternateName;
    optional(5) int overrideCode = -1;
};
```

An optional data member with a default value is considered to be set by default:

C++

```
CPtr c = new C;            // default constructor
assert(!c->alternateName); // not set
assert(c->overrideCode);   // set to default value
```

Each language mapping provides an API for resetting an optional data member to its unset condition.

See Also

- Classes
- User Exceptions
- Optional Values