Ruby Mapping for Classes

Previous Next

On this page:

- Basic Ruby Mapping for Classes
- Inheritance from Ice::Value in Ruby
- Class Data Members in Ruby
- Class Constructors in Ruby
- Class Operations in Ruby
- Value Factories in Ruby

Basic Ruby Mapping for Classes

A Slice class maps to a Ruby class with the same name. For each Slice data member, the generated class contains an instance variable and accessors to read and write it, just as for structures and exceptions. Consider the following class definition:

Slice

```
class TimeOfDay
{
    short hour;         // 0 - 23
    short minute;       // 0 - 59
    short second;       // 0 - 59
}
```

The Ruby mapping generates the following code for this definition:

Ruby

```
class TimeOfDay < ::Ice::Value
    def initialize(hour=0, minute=0, second=0)
        @hour = hour
        @minute = minute
        @second = second
    end

    attr_accessor :hours, :minutes, :seconds
end
```

There are a number of things to note about the generated code:

1. The generated class `TimeOfDay` derives from `Ice::Value`. This reflects the semantics of Slice classes in that all classes implicitly inherit from `Ice::Value`, which is the ultimate ancestor of all classes. Note that `Ice::Value` is *not* the same as `Ice::ObjectPrx`. In other words, you *cannot* pass a class where a proxy is expected and vice versa.
2. The constructor defines an instance variable for each Slice data member.

There is quite a bit to discuss here, so we will look at each item in turn.

Back to Top ^

Inheritance from `Ice::Value` in Ruby

Like interfaces, classes implicitly inherit from a common base class, `Ice::Value`. However classes inherit from `Ice::Value` instead of `Ice::ObjectPrx` (which is at the base of the inheritance hierarchy for proxies). As a result, you cannot pass a class where a proxy is expected (and vice versa) because the base types for classes and proxies are not compatible.

`Ice::Value` contains a number of methods:

Ruby

```
class Value
    def inspect
        ::Ice::__stringify(self, self.class::ICE_TYPE)
    end
    def ice_id()
        self.class::ICE_ID
    end
    def Value.ice_staticId()
        self::ICE_ID
    end
end
```

The methods behave as follows:

- `ice_id`
  This method returns the actual run-time type ID of the object. If you call `ice_id` through a reference to a base instance, the returned type id is the actual (possibly more derived) type ID of the instance.
- `ice_staticId`
  This method returns the static type ID of the class.
- `ice_preMarshal`
  If the object supports this method, the Ice run time invokes it just prior to marshaling the object's state, providing the opportunity for the object to validate its declared data members.
- `ice_postUnmarshal`
  If the object supports this method, the Ice run time invokes it after unmarshaling the object's state. An object typically defines this method when it needs to perform additional initialization using the values of its declared data members.
- `ice_getSlicedData`
  This functions returns the `SlicedData` object if the value has been sliced during un-marshaling or `nil` otherwise.

Note that neither `Ice::Value` nor the generated class override `hash` and `==`, so the default implementations apply.

Back to Top ^

Class Data Members in Ruby

By default, data members of classes are mapped exactly as for structures and exceptions: for each data member in the Slice definition, the generated class contains a corresponding instance variable and accessor methods.

Optional data members use the same mapping as required data members, but an optional data member can also be set to the marker value `Ice::Unset` to indicate that the member is unset. A well-behaved program must compare an optional data member to `Ice::Unset` before using the member's value:

Ruby

```
v = ...
if v.optionalMember == Ice::Unset
    puts "optionalMember is unset"
else
    puts "optionalMember = " + v.optionalMember
end
```

The `Ice::Unset` marker value has different semantics than `nil`. Since `nil` is a legal value for certain Slice types, the Ice run time requires a separate marker value so that it can determine whether an optional value is set. An optional value set to `nil` is considered to be set. If you need to distinguish between an unset value and a value set to `nil`, you can do so as follows:

Ruby

```
v = ...
if v.optionalMember == Ice::Unset
    puts "optionalMember is unset"
elsif v.optionalMember == nil
    puts "optionalMember is nil"
else
    puts "optionalMember = " + v.optionalMember
end
```

If you wish to restrict access to a data member, you can modify its visibility using the `protected` metadata directive. The presence of this directive causes the Slice compiler to generate the data member with protected visibility. As a result, the member can be accessed only by the class itself or by one of its subclasses. For example, the `TimeOfDay` class shown below has the `protected` metadata directive applied to each of its data members:

Slice

```
class TimeOfDay
{
    ["protected"] short hour;   // 0 - 23
    ["protected"] short minute; // 0 - 59
    ["protected"] short second; // 0 - 59
}
```

The Slice compiler produces the following generated code for this definition:

Ruby

```
class TimeOfDay < ::Ice::Value

    def initialize(hour=0, minute=0, second=0)
        @hour = hour
        @minute = minute
        @second = second
    end

    attr_accessor :hours, :minutes, :seconds
    protected :hours, :hours=
    protected :minutes, :minutes=
    protected :seconds, :seconds=
end
```

For a class in which all of the data members are protected, the metadata directive can be applied to the class itself rather than to each member individually. For example, we can rewrite the `TimeOfDay` class as follows:

Slice

```
["protected"] class TimeOfDay
{
    short hour;         // 0 - 23
    short minute;       // 0 - 59
    short second;       // 0 - 59
}
```

Class Constructors in Ruby

Classes have a constructor that assigns to each data member a default value appropriate for its type:

Data Member Type	Default Value

`string`	Empty string
`enum`	First enumerator in enumeration
`struct`	Default-constructed value
Numeric	Zero
`bool`	False
`sequence`	Null
`dictionary`	Null
`class/interface`	Null

You can also declare different default values for data members of primitive and enumerated types.

For derived classes, the constructor has one parameter for each of the base class's data members, plus one parameter for each of the derived class's data members, in base-to-derived order.

Pass the marker value `Ice::Unset` as the value of any optional data members that you wish to be unset.

Back to Top ^

Class Operations in Ruby

Deprecated Feature

Operations on classes are deprecated as of Ice 3.7. Skip this section unless you need to communicate with old applications that rely on this feature.

With the Ruby mapping, operations in classes are not mapped at all into the corresponding Ruby class. The generated Ruby class is the same whether the Slice class has operations or not.

Back to Top ^

Value Factories in Ruby

While value factories were necessary in previous Ice versions when using classes with operations (a now deprecated feature) with the Ruby mapping, value factories may be used for any kind of class and are *not* deprecated.

Value factories allow you to create classes derived from the Ruby class generated by the Slice compiler, and tell the Ice run time to create instances of these classes when unmarshaling. For example, with the following simple interface:

Slice

```
class CustomTimeOfDay extends TimeOfDay
{
    public function format() { ... prints formatted data members ... }
}
```

You then create and register a value factory for your custom class with your Ice communicator:

Ruby

```
class ValueFactory
    def create(type)
        fail unless type == M::TimeOfDay::ice_staticId()
        TimeOfDayI.new
    end
end

communicator = ...
communicator.getValueFactoryManager().add(ValueFactory.new, M::TimeOfDay::ice_staticId())
```

Back to Top ^

See Also

- Classes
- Type IDs
- Optional Data Members
- The Current Object
- Value Factories

Previous

Next