MATLAB Mapping for Classes

Previous Next

On this page:

Basic MATLAB Mapping for Classes

A Slice class is mapped to a MATLAB class with the same name. The generated class contains a public property for each Slice data member (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
    string tz;          // e.g. GMT, PST, EDT...
}
```

The Slice compiler generates the following code for this definition:

MATLAB

```
classdef TimeOfDay < Ice.Value
    properties
        hour int16
        minute int16
        second int16
        tz char
    end
    methods
        function obj = TimeOfDay(hour, minute, second, tz)
            ...
        end
        function id = ice_id(obj)
            ...
        end
        ...
    end
    methods(Static)
        function id = ice_staticId()
            ...
        end
    end
end
```

There are a several things to note about the generated code:

1. The generated class `TimeOfDay` inherits from `Ice.Value`. This means that all classes implicitly inherit from `Value`, which is the ultimate ancestor of all classes.
2. The generated class contains a public property for each Slice data member.
3. The generated class has a constructor that optionally takes one argument for each 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 MATLAB

Classes implicitly inherit from a common base class, `Value`, which is mapped to `Ice.Value`. `Value` is a very simple base class with just a few methods:

MATLAB

```
classdef (Abstract) Value < matlab.mixin.Copyable
    methods
        function ice_preMarshal(obj)
        end
        function ice_postUnmarshal(obj)
        end
        function r = ice_getSlicedData(obj)
            ...
        end
        ...
    end
    methods(Abstract)
        id = ice_id(obj)
    end
    methods(Static)
        function id = ice_staticId()
            id = '::Ice::Object'
        end
    end
end
```

`Value` derives from `matlab.mixin.Copyable`, which means subclasses are handle types and instances can be copied using the `copy` function.

The `Value` methods behave as follows:

- `ice_preMarshal`
  The Ice run time invokes this method prior to marshaling the object's state, providing the opportunity for a subclass to validate its declared data members.
- `ice_postUnmarshal`
  The Ice run time invokes this method after unmarshaling an object's state. A subclass typically overrides this method when it needs to perform additional initialization using the values of its declared data members.
- `ice_id`
  This method returns the actual run-time type ID for a class instance. 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_getSlicedData`
  This functions returns the `SlicedData` object if the value has been sliced during unmarshaling or an empty array otherwise.

Back to Top ^

Class Data Members in MATLAB

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 public property.

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 test an optional data member before using its value:

MATLAB

```
obj = ...;
if obj.optionalMember ~= Ice.Unset
    fprintf('optionalMember = %s\n', ex.optionalMember);
else
    fprintf('optionalMember is unset\n');
end
```

The `Ice.Unset` marker value has different semantics than an empty array. Since an empty array 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 an empty array is considered to be set. If you need to distinguish between an unset value and a value set to an empty array, you can do so as follows:

MATLAB

```
obj = ...;
if obj.optionalMember == Ice.Unset
    fprintf('optionalMember is unset\n');
elseif isempty(obj.optionalMember)
    fprintf('optionalMember is empty\n');
else
    fprintf('optionalMember = %s\n', ex.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 property with protected visibility. As a result, the property 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
    ["protected"] string tz;    // e.g. GMT, PST, EDT...
}
```

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

MATLAB

```
classdef TimeOfDay < Ice.Value
    properties(Access=protected)
        hour int16
        minute int16
        second int16
        tz char
    end
    ...
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
    string tz;          // e.g. GMT, PST, EDT...
}
```

Back to Top ^

Value Factories in MATLAB

Value factories allow you to create classes derived from the MATLAB classes 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

```
interface Time
{
    TimeOfDay get();
}
```

The default behavior of the Ice run time will create and return an instance of the generated `TimeOfDay` class.

If you wish, you can create your own custom derived class, and tell Ice to create and return these instances instead. For example:

MATLAB

```
classdef CustomTimeOfDay < TimeOfDay
    methods
        function format(obj)
            % prints formatted data members
        end
    end
end
```

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

MATLAB

```
function v = factory(type)
    assert(strcmp(type, TimeOfDay.ice_staticId()));
    v = CustomTimeOfDay();
end

communicator = ...;
communicator.getValueFactoryManager().add(@factory, TimeOfDay.ice_staticId());
```

Back to Top ^

Class Constructors in MATLAB

If a class declares or inherits any data members, the generated constructor accepts one parameter for each data member so that you can construct and initialize an instance in a single statement (instead of first having to construct the instance and then assign to its members). For a derived class, the constructor accepts one argument for each base class member, plus one argument for each derived class member, in base-to-derived order.

You must either call the constructor with no arguments or with arguments for all of the parameters.

Calling the constructor with no arguments assigns a default value appropriate for each member's type:

Data Member Type	Default Value
`string`	Empty string
`enum`	First enumerator in enumeration
`struct`	Default-constructed value
Numeric	Zero
`bool`	`false`
`sequence`	Empty array
`dictionary`	Instance of the mapped type
`class`	Empty array

If you wish to ensure that data members of primitive and enumerated types are initialized to specific values, you can declare default values in your Slice definition. The default constructor initializes each of these data members to its declared value instead.

Back to Top ^

See Also

- Classes
- Class Inheritance
- Type IDs
- Value Factories

Previous

Next