MATLAB Mapping for Exceptions

Previous Next

On this page:

MATLAB Mapping for User Exceptions

Here is a fragment of the Slice definition for our world time server once more:

Slice

```
exception GenericError
{
    string reason;
}
exception BadTimeVal extends GenericError {}
exception BadZoneName extends GenericError {}
```

These exception definitions map as follows:

MATLAB

```
classdef GenericError < Ice.UserException
    properties
        reason char
    end
    methods
        function obj = GenericError(ice_exid, ice_exmsg, reason)
            ...
        end
        function id = ice_id(obj)
            ...
        end
    end
    ...
end

classdef BadTimeVal < GenericError
    methods
        function obj = BadTimeVal(ice_exid, ice_exmsg, reason)
            ...
        end
        function id = ice_id(obj)
            ...
        end
    end
    ...
end

classdef BadZoneName < GenericError
    methods
        function obj = BadZoneName(ice_exid, ice_exmsg, reason)
            ...
        end
        function id = ice_id(obj)
            ...
        end
    end
    ...
end
```

Each Slice exception is mapped to a MATLAB class with the same name. For each data member, the corresponding class contains a public property. (Obviously, because `BadTimeVal` and `BadZoneName` do not have members, the generated classes for these exceptions also do not have properties.)

The inheritance structure of the Slice exceptions is preserved for the generated classes, so `BadTimeVal` and `BadZoneName` inherit from `GenericError`.

Each exception also defines an `ice_id` method, which returns the Slice type ID of the exception.

All user exceptions are derived from the base class `UserException`. This allows you to handle all user exceptions generically by testing whether an instance is-a `UserException`. `UserException`, in turn, derives from `Exception`, which derives from MATLAB's native `MException` class.

Note that the generated exception classes contain other methods that are not shown. However, those methods are internal to the MATLAB mapping and are not meant to be called by application code.

Here's an example that shows how we could handle these exceptions:

MATLAB

```
try
    % ...
catch ex
    if isa(ex, 'BadZoneName')
        % handle BadZoneName
    elseif isa(ex, 'BadTimeVal')
        % handle BadTimeVal
    elseif isa(ex, 'GenericError')
        % handle GenericError
    else
        % Allow any other exception to propagate
        rethrow(ex);
    end
end
```

Back to Top ^

Constructing a User Exception

The first two arguments for every exception constructor are an identifier and a message; these arguments are passed up the inheritance hierarchy to the `MException` class. You can pass empty strings for these arguments and the constructor will supply default values.

If an exception declares or inherits any data members, the constructor accepts one additional 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 exception, the constructor accepts one argument for each base exception member, plus one argument for each derived exception 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 ^

Optional Data Members

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

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

```
try
    ...
catch ex
    if ex.optionalMember == Ice.Unset
        fprintf('optionalMember is unset\n');
    elseif isempty(ex.optionalMember)
        fprintf('optionalMember is empty\n');
    else
        fprintf('optionalMember = %s\n', ex.optionalMember);
    end
end
```

Back to Top ^

MATLAB Mapping for Run-Time Exceptions

The Ice run time throws run-time exceptions for a number of pre-defined error conditions. All run-time exceptions directly or indirectly derive from `LocalException` (which, in turn, derives indirectly from `MException`).

Recall the inheritance diagram for user and run-time exceptions. By testing exceptions at the appropriate point in the hierarchy, you can handle exceptions according to the category of error they indicate:

- `LocalException`
  This is the root of the inheritance tree for run-time exceptions.
- `UserException`
  This is the root of the inheritance tree for user exceptions.
- `TimeoutException`
  This is the base exception for both operation-invocation and connection-establishment timeouts.
- `ConnectTimeoutException`
  This exception is raised when the initial attempt to establish a connection to a server times out.

For example, a `ConnectTimeoutException` can be handled as `ConnectTimeoutException`, `TimeoutException`, `LocalException`, or `MException`.

You will probably have little need to test run-time exceptions for their most-derived type and instead test them as `LocalException`; the fine-grained error handling offered by the remainder of the hierarchy is of interest mainly in the implementation of the Ice run time. Exceptions to this rule are the exceptions related to facet and object life cycles, which you may want to handle explicitly. These exceptions are `FacetNotExistException` and `ObjectNotExistException`, respectively.

Back to Top ^

See Also

- User Exceptions
- Run-Time Exceptions

Previous

Next