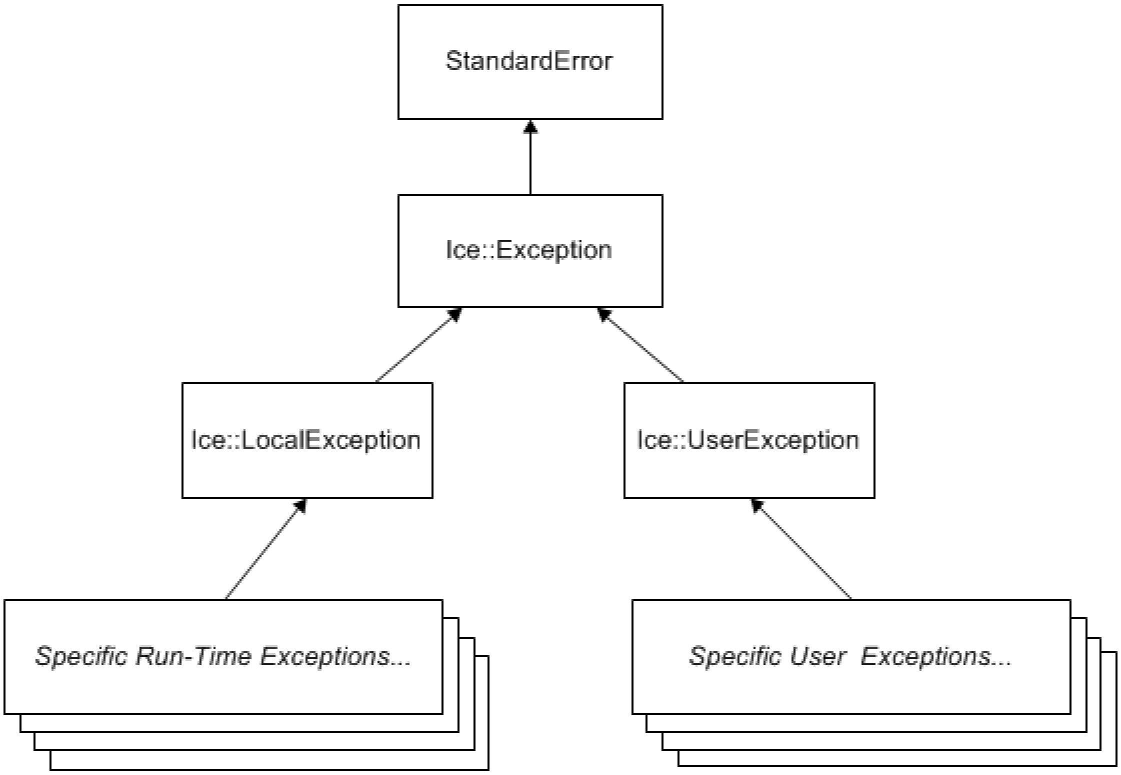
Ruby Mapping for Exceptions

On this page:

Inheritance Hierarchy for Exceptions in Ruby

The mapping for exceptions is based on the inheritance hierarchy shown below:



Inheritance structure for Ice exceptions.

The ancestor of all exceptions is `StandardError`, from which `Ice::Exception` is derived. `Ice::LocalException` and `Ice::UserException` are derived from `Ice::Exception` and form the base for all run-time and user exceptions.

Ruby 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 to the abbreviated Ruby class definitions shown below:

Ruby

```
class GenericError < Ice::UserException
    def initialize(reason='')

    def to_s

    def inspect

    attr_accessor :reason
end

class BadTimeVal < GenericError
    def initialize(reason='')

    def to_s

    def inspect
end

class BadZoneName < GenericError
    def initialize(reason='')

    def to_s

    def inspect
end
```

Each Slice exception is mapped to a Ruby class with the same name. The inheritance structure of the Slice exceptions is preserved for the generated classes, so `BadTimeVal` and `BadZoneName` inherit from `GenericError`.

Each exception member corresponds to an instance variable of the instance, which the constructor initializes to a default value appropriate for its type. You can also declare different default values for members of primitive and enumerated types. For derived exceptions, the constructor has one parameter for each of the base exception's data members, plus one parameter for each of the derived exception's data members, in base-to-derived order. As an example, although `BadTimeVal` and `BadZoneName` do not declare data members, their constructors still accept a value for the inherited data member `reason` in order to pass it to the constructor of the base exception `GenericError`. The generated class defines an accessor for each data member to read and write its value.

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

```
begin
    ...
rescue => ex
    if ex.optionalMember == Ice::Unset
        puts "optionalMember is unset"
    else
        puts "optionalMember = " + ex.optionalMember
    end
end
```

Each exception also defines the standard methods `to_s` and `inspect` to return the name of the exception and a stringified representation of the exception and its members, respectively.

All user exceptions are derived from the base class `Ice::UserException`. This allows you to catch all user exceptions generically by installing a handler for `Ice::UserException`. Similarly, you can catch all Ice run-time exceptions with a handler for `Ice::LocalException`, and you can catch all Ice exceptions with a handler for `Ice::Exception`.

Ruby 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 `Ice::LocalException` (which, in turn, derives from `Ice::Exception`).

By catching exceptions at the appropriate point in the inheritance hierarchy, you can handle exceptions according to the category of error they indicate:

- `Ice::LocalException`
  This is the root of the inheritance tree for run-time exceptions.

- `Ice::UserException`
  This is the root of the inheritance tree for user exceptions.

- `Ice::TimeoutException`
  This is the base exception for both operation-invocation and connection-establishment timeouts.

- `Ice::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 `Exception`.

You will probably have little need to catch run-time exceptions as their most-derived type and instead catch 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 catch explicitly. These exceptions are `FacetNotExistException` and `ObjectNotExistException`, respectively.

See Also

- User Exceptions
- Run-Time Exceptions
- Ruby Mapping for Identifiers
- Ruby Mapping for Modules
- Ruby Mapping for Built-In Types
- Ruby Mapping for Enumerations
- Ruby Mapping for Structures
- Ruby Mapping for Sequences
- Ruby Mapping for Dictionaries
- Ruby Mapping for Constants
- Optional Data Members
- Facets and Versioning
- Object Life Cycle