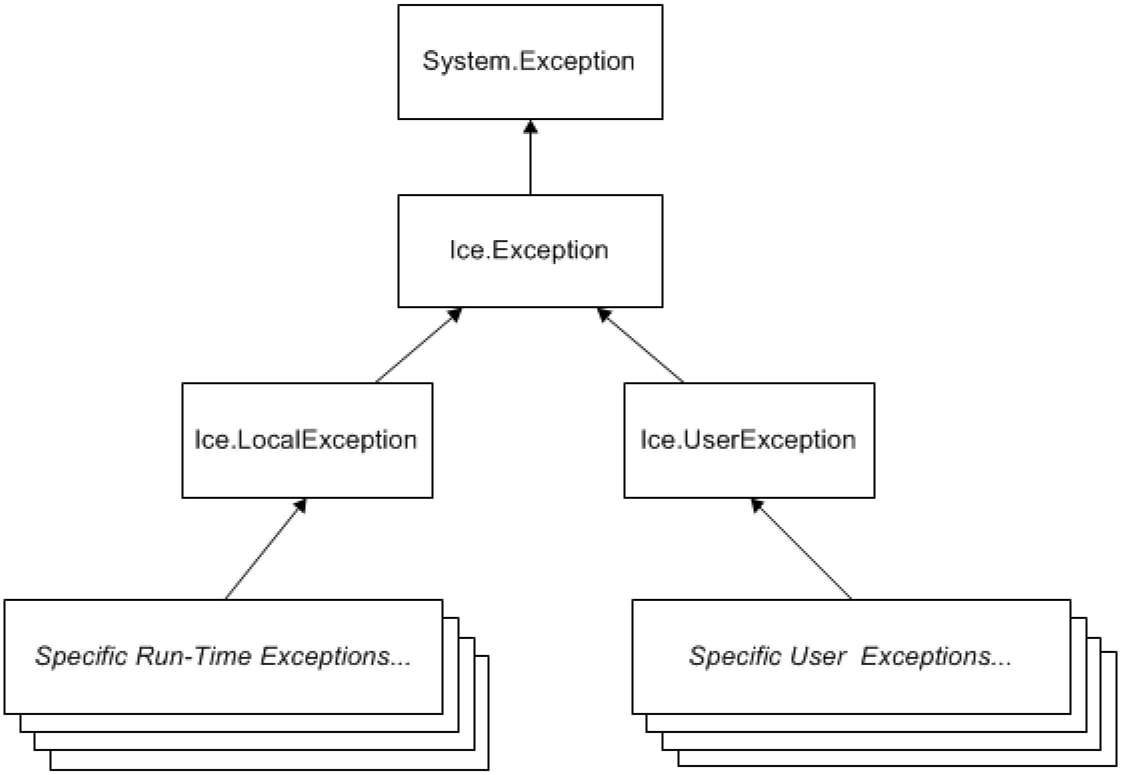
C-Sharp Mapping for Exceptions

On this page:

Inheritance Hierarchy for Exceptions in C#

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



Inheritance structure for exceptions.

The ancestor of all exceptions is `System.Exception`. Derived from that is `Ice.Exception`, which provides the definitions of a number of constructors. `Ice.LocalException` and `Ice.UserException` are derived from `Ice.Exception` and form the base of all run-time and user exceptions, respectively.

The constructors defined in `Ice.Exception` have the following signatures:

C#

```
public abstract class Exception : System.Exception
{
    public Exception();
    public Exception(System.Exception ex);
}
```

Each concrete derived exception class implements these constructors. The second constructor initializes the `InnerException` property of `System.Exception`. (Both constructors set the `Message` property to the empty string.)

C# 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:

C#

```
public partial class GenericError : Ice.UserException
{
    public string reason;

    public GenericError();
    public GenericError(System.Exception ex__);
    public GenericError(string reason);
    public GenericError(string reason, System.Exception ex__);

    // GetHashCode and comparison methods defined here,
    // as well as mapping-internal methods.
}

public partial class BadTimeVal : M.GenericError
{
    public BadTimeVal();
    public BadTimeVal(System.Exception ex__);
    public BadTimeVal(string reason);
    public BadTimeVal(string reason, System.Exception ex__);

    // GetHashCode and comparison methods defined here,
    // as well as mapping-internal methods.
}

public partial class BadZoneName : M.GenericError
{
    public BadZoneName();
    public BadZoneName(System.Exception ex__);
    public BadZoneName(string reason);
    public BadZoneName(string reason, System.Exception ex__);

    // GetHashCode and comparison methods defined here,
    // as well as mapping-internal methods.
}
```

Each Slice exception is mapped to a C# partial class with the same name. For each exception member, the corresponding class contains a public data member. (Obviously, because `BadTimeVal` and `BadZoneName` do not have members, the generated classes for these exceptions also do not have members.) Optional data members are mapped to instances of the `Ice.Optional` type.

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

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`.

All exceptions provide the usual `GetHashCode` and `Equals` methods, as well as the `==` and `!=` comparison operators.

The generated exception classes also contain other member functions that are not shown here; these member functions are internal to the C# mapping and are not meant to be called by application code.

C# Default Constructors for User Exceptions

Exceptions have a default constructor that default-constructs each data member. This means members of primitive type are initialized to the equivalent of zero, and members of reference type are initialized to null. Note that applications must always explicitly initialize a member whose type is a class-mapped structure because the Ice run time does not accept null as a legal value for these types.

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.

An exception also provides a constructor that accepts one parameter for each data member so that you can construct and initialize a class instance in a single statement (instead of first having to construct the instance and then assign to its members). For a derived exception, this constructor accepts one argument for each base exception member, plus one argument for each derived exception member, in base-to-derived order. For each optional data member, the constructor accepts an `Ice.Optional` parameter of the appropriate type.

C# 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`).

`Ice.LocalException` implements a `Clone` method that is inherited by its derived exceptions, so you can make memberwise shallow copies of exceptions.

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

- `Ice.Exception`
  This is the root of the inheritance tree for both run-time and user exceptions.

- `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
- C-Sharp Mapping for Identifiers
- C-Sharp Mapping for Modules
- C-Sharp Mapping for Built-In Types
- C-Sharp Mapping for Enumerations
- C-Sharp Mapping for Structures
- C-Sharp Mapping for Sequences
- C-Sharp Mapping for Dictionaries
- C-Sharp Collection Comparison
- C-Sharp Mapping for Constants
- C-Sharp Mapping for Optional Values
- Facets and Versioning
- Object Life Cycle