

# Python Mapping for Exceptions

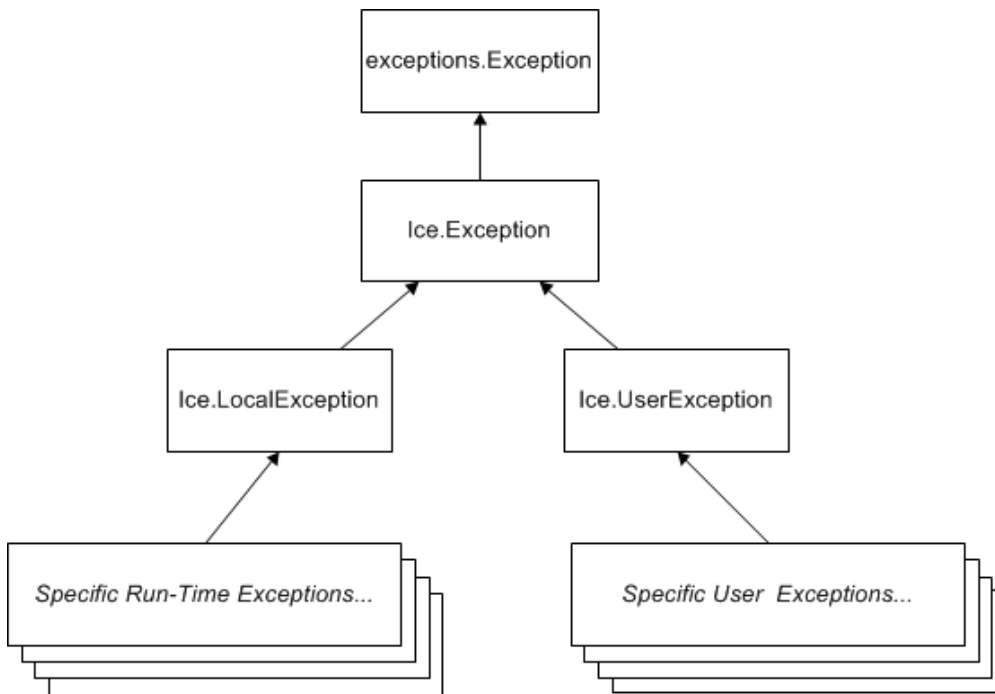


On this page:

- [Inheritance Hierarchy for Exceptions in Python](#)
- [Python Mapping for User Exceptions](#)
  - [Optional Data Members](#)
- [Python Mapping for Run-Time Exceptions](#)

## Inheritance Hierarchy for Exceptions in Python

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



*Inheritance structure for Ice exceptions.*

The ancestor of all exceptions is `exceptions.Exception`, 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.

## Python 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:

## Python

```
class GenericError(Ice.UserException):
    def __init__(self, reason=''):
        self.reason = reason

    def ice_id(self):
        # ...

    def __str__(self):
        # ...

class BadTimeVal(GenericError):
    def __init__(self, reason=''):
        GenericError.__init__(self, reason)

    def ice_id(self):
        # ...

    def __str__(self):
        # ...

class BadZoneName(GenericError):
    def __init__(self, reason=''):
        GenericError.__init__(self, reason)

    def ice_id(self):
        # ...

    def __str__(self):
        # ...
```

Each Slice exception is mapped to a Python 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 attribute of the instance, which the constructor initializes to 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	None
dictionary	None
class/interface	None

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`.

Each exception also defines the `ice_id` method that returns the Slice type ID of the exception, and the special method `__str__` to return a stringified representation of the exception and its members.

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`.

[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:

#### Python

```
try:
    ...
except ex:
    if ex.optionalMember:
        print("optionalMember = " + str(ex.optionalMember))
    else:
        print("optionalMember is unset")
```

The `Ice.Unset` marker value has different semantics than `None`. Since `None` 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 `None` is considered to be set. If you need to distinguish between an unset value and a value set to `None`, you can do so as follows:

#### Python

```
try:
    ...
except ex:
    if ex.optionalMember is Ice.Unset:
        print("optionalMember is unset")
    elif ex.optionalMember is None:
        print("optionalMember is None")
    else:
        print("optionalMember = " + str(ex.optionalMember))
```

[Back to Top ^](#)

## Python 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.

[Back to Top ^](#)

#### See Also

- [User Exceptions](#)
- [Run-Time Exceptions](#)
- [Python Mapping for Identifiers](#)
- [Python Mapping for Modules](#)
- [Python Mapping for Built-In Types](#)
- [Python Mapping for Enumerations](#)

- Python Mapping for Structures
- Python Mapping for Sequences
- Python Mapping for Dictionaries
- Python Mapping for Constants
- Optional Data Members
- Versioning
- Object Life Cycle

