Run-Time Exceptions

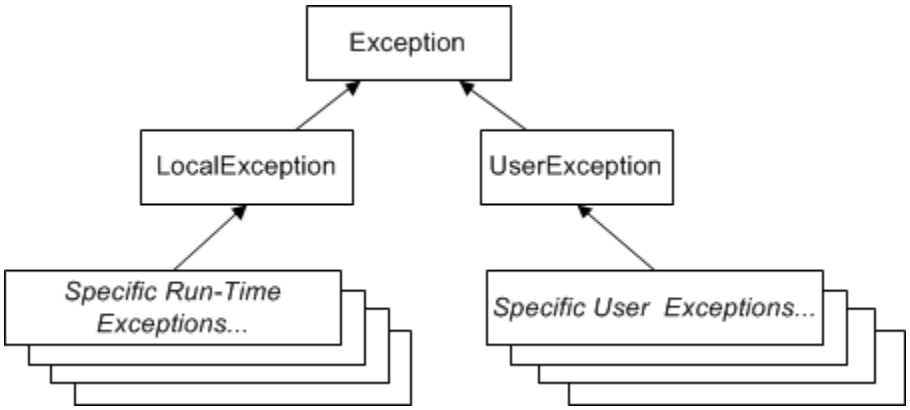
In addition to any user exceptions that are listed in an operation's exception specification, an operation can also throw Ice *run-time exceptions*. Run-time exceptions are predefined exceptions that indicate platform-related run-time errors. For example, if a networking error interrupts communication between client and server, the client is informed of this by a run-time exception, such as `ConnectTimeoutException` or `SocketException`.

The exception specification of an operation must not list any run-time exceptions. (It is understood that all operations can raise run-time exceptions and you are not allowed to restate that.)

On this page:

Inheritance Hierarchy for Exceptions

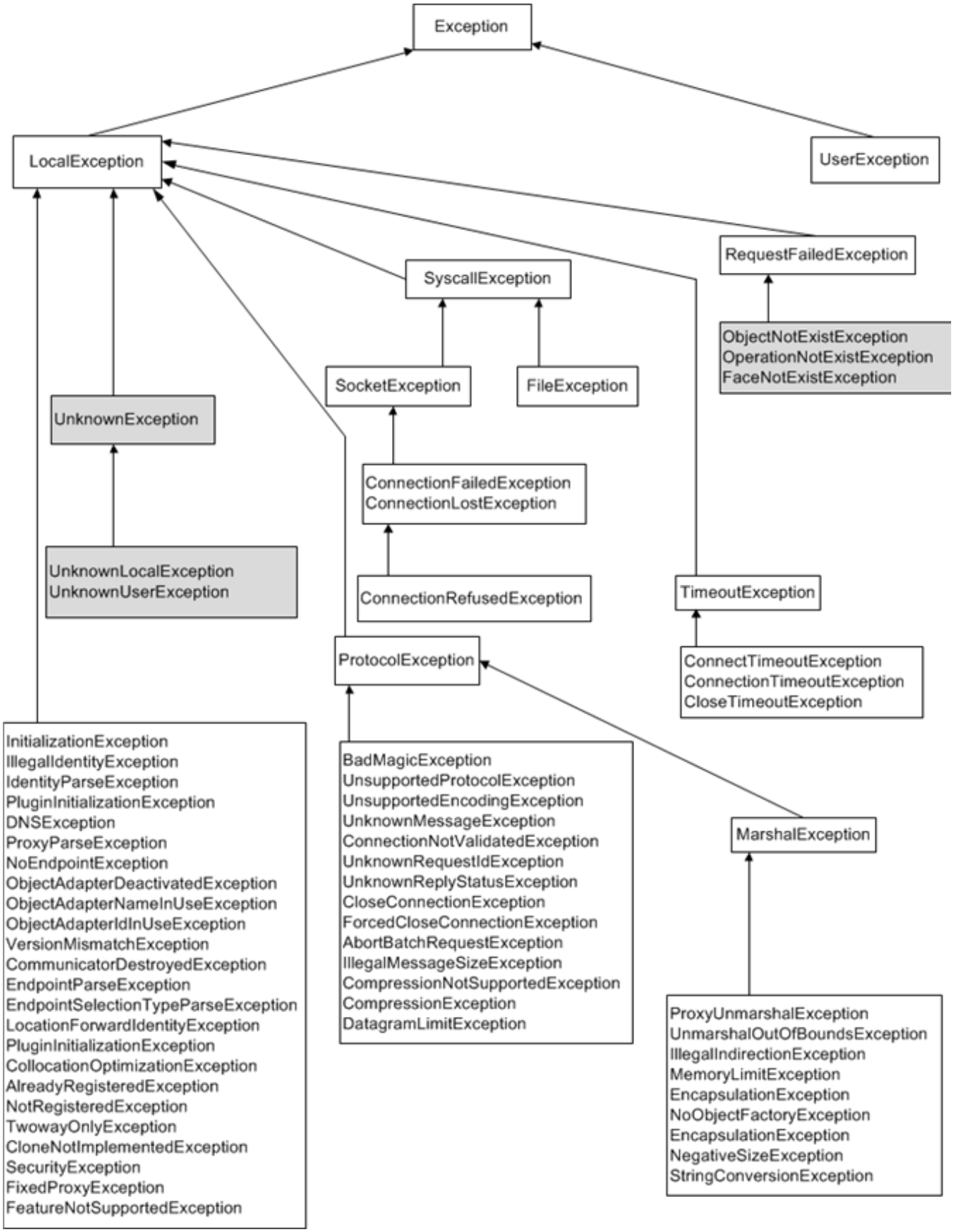
All the Ice run-time and user exceptions are arranged in an inheritance hierarchy, as shown below:



Inheritance structure for exceptions.

`Ice::Exception` is at the root of the inheritance hierarchy. Derived from that are the (abstract) types `Ice::LocalException` and `Ice::UserException`. In turn, all run-time exceptions are derived from `Ice::LocalException`, and all user exceptions are derived from `Ice::UserException`.

This figures shows the complete hierarchy of the Ice run-time exceptions:



Ice run-time exception hierarchy. (Shaded exceptions can be sent by the server.)

We use the Unified Modeling Language (UML) for the object model diagrams (see [1] and [2] for details).

Note that Ice run-time exception hierarchy groups several exceptions into a single box to save space (which, strictly, is incorrect UML syntax). Also note that some run-time exceptions have data members, which, for brevity, we have omitted in the Ice run-time exception hierarchy. These data members provide additional information about the precise cause of an error.

Many of the run-time exceptions have self-explanatory names, such as `MemoryLimitException`. Others indicate problems in the Ice run time, such as `EncapsulationException`. Still others can arise only through application programming errors, such as `TwowayOnlyException`. In practice, you will likely never see most of these exceptions. However, there are a few run-time exceptions you will encounter and whose meaning you should know.

Local Versus Remote Exceptions

Common Exceptions

Most error conditions are detected on the client side. For example, if an attempt to contact a server fails, the client-side run time raises a `ConnectTimeoutException`. However, there are three specific error conditions (shown as shaded in the Ice run-time exception hierarchy diagram) that are detected by the server and made known explicitly to the client-side run time via the Ice protocol: `ObjectNotExistException`, `FacetNotExistException`, and `OperationNotExistException`.

`ObjectNotExistException`

This exception indicates that a request was delivered to the server but the server could not locate a servant with the identity that is embedded in the proxy. In other words, the server could not find an object to dispatch the request to.

An `ObjectNotExistException` is a death certificate: it indicates that the target object in the server does not exist.

The Ice run time raises `ObjectNotExistException` only if there are no facets in existence with a matching identity; otherwise, it raises `FacetNotExistException`.

Most likely, this is the case because the object existed some time in the past and has since been destroyed, but the same exception is also raised if a client uses a proxy with the identity of an object that has never been created. If you receive this exception, you are expected to clean up whatever resources you might have allocated that relate to the specific object for which you receive this exception.

`FacetNotExistException`

The client attempted to contact a non-existent facets of an object, that is, the server has at least one servant with the given identity, but no servant with a matching facet name.

`OperationNotExistException`

This exception is raised if the server could locate an object with the correct identity but, on attempting to dispatch the client's operation invocation, the server found that the target object does not have such an operation. You will see this exception in only two cases:

- You have used an unchecked down-cast on a proxy of the incorrect type.
- Client and server have been built with Slice definitions for an interface that disagree with each other, that is, the client was built with an interface definition for the object that indicates that an operation exists, but the server was built with a different version of the interface definition in which the operation is absent.

Unknown Exceptions

Any error condition on the server side that is not described by one of the three preceding exceptions is made known to the client as one of three generic exceptions (shown as shaded in the Ice run-time exception hierarchy figure diagram): `UnknownUserException`, `UnknownLocalException`, or `UnknownException`.

`UnknownUserException`

This exception indicates that an operation implementation has thrown a Slice exception that is not declared in the operation's exception specification (and is not derived from one of the exceptions in the operation's exception specification).

`UnknownLocalException`

If an operation implementation raises a run-time exception other than `ObjectNotExistException`, `FacetNotExistException`, or `OperationNotExistException` (such as a `NotRegisteredException`), the client receives an `UnknownLocalException`. In other words, the Ice protocol does not transmit the exact exception that was encountered in the server, but simply returns a bit to the client in the reply to indicate that the server encountered a run-time exception.

A common cause for a client receiving an `UnknownLocalException` is failure to catch and handle all exceptions in the server. For example, if the implementation of an operation encounters an exception it does not handle, the exception propagates all the way up the call stack until the stack is unwound to the point where the Ice run time invoked the operation. The Ice run time catches all Ice exceptions that "escape" from an operation invocation and returns them to the client as an `UnknownLocalException`.

`UnknownException`

An operation has thrown a non-Ice exception. For example, if the operation in the server throws a C++ exception, such as a `char*`, or a Java exception, such as a `ClassCastException`, the client receives an `UnknownException`.

All other run-time exceptions (not shaded in the Ice run-time exception hierarchy) are detected by the client-side run time and are raised locally.

It is possible for the implementation of an operation to throw Ice run-time exceptions (as well as user exceptions). For example, if a client holds a proxy to an object that no longer exists in the server, your server application code is required to throw an `ObjectNotExistException`. If you do throw run-time exceptions from your application code, you should take care to throw a run-time exception only if appropriate, that is, do not use run-time exceptions to indicate something that really should be a user exception. Doing so can be very confusing to the client: if the application "hijacks" some run-time exceptions for its own purposes, the client can no longer decide whether the exception was thrown by the Ice run time or by the server application code. This can make debugging very difficult.

See Also

- User Exceptions
- Interfaces, Operations, and Exceptions
- Operations
- Proxies
- Interface Inheritance
- Facets and Versioning

References

1. Booch, G., et al. 1998. *Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.
2. Object Management Group. 2001. *Unified Modeling Language Specification*. Framingham, MA: Object Management Group.