Raising Exceptions in C++

To throw an exception from an operation implementation, you simply instantiate the exception, initialize it, and throw it. For example:

C++

```
void
Filesystem::FileI::write(const Filesystem::Lines& text, const Ice::Current&)
{
    // Try to write the file contents here...
    // Assume we are out of space...
    if (error) {
        Filesystem::GenericError e;
        e.reason = "file too large";
        throw e;
    }
};
```

No memory management issues arise in the presence of exceptions.

Note that the Slice compiler never generates exception specifications for operations, regardless of whether the corresponding Slice operation definition has an exception specification or not. This is deliberate: C++ exception specifications do not add any value and are therefore not used by the Ice C++ mapping [1].

If you throw an arbitrary C++ exception (such as an `int` or other unexpected type), the Ice run time catches the exception and then returns an `UnknownException` to the client. Similarly, if you throw an "impossible" user exception (a user exception that is not listed in the exception specification of the operation), the client receives an `UnknownUserException`.

If you throw a run-time exception, such as `MemoryLimitException`, the client receives an `UnknownLocalException`. For that reason, you should never throw system exceptions from operation implementations. If you do, all the client will see is an `UnknownLocalException`, which does not tell the client anything useful.

Three run-time exceptions are treated specially and not changed to `UnknownLocalException` when returned to the client: `ObjectNotExistException`, `OperationNotExistException`, and `FacetNotExistException`.

See Also

- Run-Time Exceptions
- C++ Mapping for Exceptions
- Server-Side C++ Mapping for Interfaces
- Parameter Passing in C++

References

1. Sutter, H. 2002. A Pragmatic Look at Exception Specifications. C/C++ Users Journal 20 (7): 59-64.