Intercepting User Exception Insertion and Extraction in C-Sharp

Inserting a User Exception in C#

As in the case of Ice objects, a Dynamic Ice application may represent user exceptions in a native format that is not directly compatible with the Ice API. If the application needs to raise such a user exception to the Ice run time, the exception must be wrapped in a subclass of `Ice.UserException`. The Dynamic Ice API provides a class to simplify this process:

C#

```
namespace Ice
{
    public abstract class UserExceptionWriter : UserException
    {
        public UserExceptionWriter(Communicator communicator);

        public abstract void write(OutputStream os);

        // ...
    }
}
```

A subclass of `UserExceptionWriter` is responsible for supplying a communicator to the constructor, and for implementing the following methods:

- `void write(OutputStream os)`
  This method is invoked when the Ice run time is ready to marshal the exception. The subclass must marshal the exception using the encoding rules for exceptions.

Extracting a User Exception in C#

An application extracts a user exception by calling one of two versions of the `throwException` method defined in the `InputStream` class:

C#

```
namespace Ice {
    public interface InputStream {
        void throwException();
        void throwException(UserExceptionReaderFactory factory);

        // ...
    }
}
```

The version without any arguments attempts to locate and throw a C# implementation of the encoded exception using classes generated by the Slice-to-C# compiler.

If your goal is to create an exception in another type system, such as a native PHP exception object, you must call the second version of `throwException` and pass an implementation of `UserExceptionReaderFactory`:

C#

```
namespace Ice {
    public interface UserExceptionReaderFactory {
        void createAndThrow(string typeId);
    }
}
```

As the stream iterates over slices of an exception from most-derived to least-derived, it invokes `createAndThrow` passing the type ID of each slice, giving the application an opportunity to raise an instance of `UserExceptionReader`:

C#

```
namespace Ice {
    public abstract class UserExceptionReader : UserException {
        protected UserExceptionReader(Communicator communicator);

        public abstract void read(InputStream is);

        public abstract string ice_name();

        protected Communicator _communicator;
    }
}
```

Subclasses of `UserExceptionReader` must implement the abstract functions. In particular, the implementation of `read` must call `InputStream.startException`, unmarshal the remaining slices, and then call `InputStream.endException`.

See Also

- Intercepting Object Insertion and Extraction in C-Sharp
- Data Encoding for Exceptions