Client-Side Slice-to-C++ Mapping

The client-side Slice-to-C++ mapping defines how Slice data types are translated to C++ types, and how clients invoke operations, pass parameters, and handle errors. Much of the C++ mapping is intuitive. For example, Slice sequences map to STL vectors, so there is essentially nothing new you have to learn in order to use Slice sequences in C++.

The rules that make up the C++ mapping are simple and regular. In particular, the mapping is free from the potential pitfalls of memory management: all types are self-managed and automatically clean up when instances go out of scope. This means that you cannot accidentally introduce a memory leak by, for example, ignoring the return value of an operation invocation or forgetting to deallocate memory that was allocated by a called operation.

The C++ mapping is fully thread-safe. For example, the reference counting mechanism for classes is interlocked against parallel access, so reference counts cannot be corrupted if a class instance is shared among a number of threads. Obviously, you must still synchronize access to data from different threads. For example, if you have two threads sharing a sequence, you cannot safely have one thread insert into the sequence while another thread is iterating over the sequence. However, you only need to concern yourself with concurrent access to your own data — the Ice run time itself is fully thread safe, and none of the Ice API calls require you to acquire or release a lock before you safely can make the call.

Much of what appears in this chapter is reference material. We suggest that you skim the material on the initial reading and refer back to specific sections as needed. However, we recommend that you read at least the mappings for exceptions, interfaces, and operations in detail because these sections cover how to call operations from a client, pass parameters, and handle exceptions.

In order to use the C++ mapping, you should need no more than the Slice definition of your application and knowledge of the C++ mapping rules. In particular, looking through the generated header files in order to discern how to use the C++ mapping is likely to be confusing because the header files are not necessarily meant for human consumption and, occasionally, contain various cryptic constructs to deal with operating system and compiler idiosyncrasies. Of course, occasionally, you may want to refer to a header file to confirm a detail of the mapping, but we recommend that you otherwise use the material presented here to see how to write your client-side code.

The `Ice` Namespace
All of the APIs for the Ice run time are nested in the `Ice` namespace, to avoid clashes with definitions for other libraries or applications. Some of the contents of the `Ice` namespace are generated from Slice definitions; other parts of the `Ice` namespace provide special-purpose definitions that do not have a corresponding Slice definition. We will incrementally cover the contents of the `Ice` namespace throughout the remainder of the manual.

Topics

- C++ Mapping for Identifiers
- C++ Mapping for Modules
- C++ Mapping for Built-In Types
- C++ Mapping for Enumerations
- C++ Mapping for Structures
- C++ Mapping for Sequences
- C++ Mapping for Dictionaries
- C++ Mapping for Constants
- C++ Mapping for Exceptions
- C++ Mapping for Interfaces
- C++ Mapping for Operations
- C++ Mapping for Classes
- C++ Mapping for Optional Values
- Smart Pointers for Classes
- Asynchronous Method Invocation (AMI) in C++
- slice2cpp Command-Line Options
- Using Slice Checksums in C++
- Example of a File System Client in C++