C++ Mapping for Built-In Types

On this page:

Mapping of Slice Built-In Types to C++ Types

The Slice built-in types are mapped to C++ types as shown in this table:

Slice	C++
bool	`bool`
byte	`Ice::Byte`
short	`Ice::Short`
int	`Ice::Int`
long	`Ice::Long`
float	`Ice::Float`
double	`Ice::Double`
string	`std::string`

Slice `bool` and `string` map to C++ `bool` and `std::string`. The remaining built-in Slice types map to C++ type definitions instead of C++ native types. This allows the Ice run time to provide a definition as appropriate for each target architecture. (For example, `Ice::Int` might be defined as `long` on one architecture and as `int` on another.)

> Note that `Ice::Byte` is a typedef for `unsigned char`. This guarantees that byte values are always in the range 0..255.

All the basic types are guaranteed to be distinct C++ types, that is, you can safely overload functions that differ in only the types listed in the table above.

Alternative String Mapping for C++

You can use a metadata directive, `["cpp:type:wstring"]`, to map strings to C++ `std::wstring`. This is useful for applications that use languages with alphabets that cannot be represented in 8?bit characters. The metadata directive can be applied to any Slice construct. For containers (such as modules, interfaces, or structures), the metadata directive applies to all strings within the container. A corresponding metadata directive, `["cpp:type:string"]`, can be used to selectively override the mapping defined by the enclosing container. For example:

Slice

```
["cpp:type:wstring"]
struct S1 {
    string x;             // Maps to std::wstring
    ["cpp:type:wstring"]
    string y;             // Maps to std::wstring
    ["cpp:type:string"]
    string z;             // Maps to std::string
};

struct S2 {
    string x;             // Maps to std::string
    ["cpp:type:string"]
    string y;             // Maps to std::string
    ["cpp:type:wstring"]
    string z;             // Maps to std::wstring
};
```

With these metadata directives, the strings are mapped as indicated by the comments. By default, narrow strings are encoded as UTF?8, and wide strings use Unicode in an encoding that is appropriate for the platform on which the application executes. You can override the encoding for narrow and wide strings by registering a string converter with the Ice run time.

See Also

- Basic Types
- C++ Mapping for Identifiers
- C++ Mapping for Modules
- 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++ Strings and Character Encoding