Enumerations

A Slice enumerated type definition looks like the C++ version:

Slice

```
enum Fruit { Apple, Pear, Orange };
```

This definition introduces a type named `Fruit` that becomes a new type in its own right. Slice does not define how ordinal values are assigned to enumerators. For example, you cannot assume that the enumerator `Orange` will have the value 2 in different implementation languages. Slice guarantees only that the ordinal values of enumerators increase from left to right, so `Apple` compares less than `Pear` in all implementation languages.

Unlike C++, Slice does not permit you to control the ordinal values of enumerators (because many implementation languages do not support such a feature):

Slice

```
enum Fruit { Apple = 0, Pear = 7, Orange = 2 }; // Syntax error
```

In practice, you do not care about the values used for enumerators as long as you do not transmit the *ordinal value* of an enumerator between address spaces. For example, sending the value 0 to a server to mean `Apple` can cause problems because the server may not use 0 to represent `Apple`. Instead, simply send the value `Apple` itself. If `Apple` is represented by a different ordinal value in the receiving address space, that value will be appropriately translated by the Ice run time.

As with C++, Slice enumerators enter the enclosing namespace, so the following is illegal:

Slice

```
enum Fruit { Apple, Pear, Orange };
enum ComputerBrands { Apple, IBM, Sun, HP };    // Apple redefined
```

Slice does not permit empty enumerations.

See Also

- Structures
- Sequences
- Dictionaries
- Constants and Literals