

# Enumerations



## Enumeration Syntax and Semantics

A Slice enumerated type definition looks identical to C++:

### Slice

```
module M
{
    enum Fruit { Apple, Pear, Orange }
}
```

This definition introduces a type named `Fruit` that becomes a new type in its own right. Slice guarantees that the values of enumerators increase from left to right, so `Apple` compares less than `Pear` in every language mapping. By default, the first enumerator has a value of zero, with sequentially increasing values for subsequent enumerators.

A Slice enum type introduces a new namespace scope, so the following is legal:

### Slice

```
module M
{
    enum Fruit { Apple, Pear, Orange }
    enum ComputerBrands { Apple, Dell, HP, Lenovo }
}
```

The example below shows how to refer to an enumerator from a different scope:

### Slice

```
module M
{
    enum Color { Red, Green, Blue }
}
module N
{
    struct Pixel
    {
        M::Color c = Blue;
    }
}
```

Slice does not permit empty enumerations.



In Ice releases prior to Ice 3.7, an enum type did not create a new namespace and its enumerators were in the same namespace as the enum type itself. With these releases, you had to select longer enumerator names to avoid a naming clash.

[Back to Top ^](#)

## Custom Enumerator Values

Slice also permits you to assign custom values to enumerators:

## Slice

```
const int PearValue = 7;
enum Fruit { Apple = 0, Pear = PearValue, Orange }
```

Custom values must be unique and non-negative, and may refer to Slice constants of integer types. If no custom value is specified for an enumerator, its value is one greater than the enumerator that immediately precedes it. In the example above, `Orange` has the value 8.

The maximum value for an enumerator value is the same as the maximum value for `int`,  $2^{31} - 1$ .

Slice does not require custom enumerator values to be declared in increasing order:

## Slice

```
enum Fruit { Apple = 5, Pear = 3, Orange = 1 }    // Legal
```

Note however that when there is an inconsistency between the declaration order and the numerical order of the enumerators, the behavior of comparison operations may vary between language mappings.



For an application that is still using version 1.0 of the [Ice encoding](#), changing the definition of an enumerated type may break backward compatibility with existing applications. For more information, please refer to the [encoding rules](#) for enumerators.

[Back to Top ^](#)

## See Also

- [Structures](#)
- [Sequences](#)
- [Dictionaries](#)
- [Constants and Literals](#)



Previous



Next