

Simple Classes



A Slice class definition is similar to a structure definition, but uses the `class` keyword. For example:

Slice

```
module M
{
    class TimeOfDay
    {
        short hour;           // 0 - 23
        short minute;         // 0 - 59
        short second;         // 0 - 59
    }
}
```

Apart from the keyword `class`, this definition is identical to the [structure](#) example. You can use a Slice class wherever you can use a Slice structure (but, as we will see shortly, for performance reasons, you should not use a class where a structure is sufficient). Unlike structures, classes can be empty:

Slice

```
class EmptyClass {} // OK
struct EmptyStruct {} // Error
```

Much the same design considerations as for [empty interfaces](#) apply to empty classes: you should at least stop and rethink your approach before committing yourself to an empty class.

A class can define any number of data members, including [optional data members](#). You can also specify a default value for a data member if its type is one of the following:

- An [integral](#) type (byte, short, int, long)
- A [floating point](#) type (float or double)
- [string](#)
- [bool](#)
- [enum](#)

For example:

Slice

```
class Location
{
    string name;
    Point pt;
    bool display = true;
    string source = "GPS";
}
```

The legal syntax for literal values is the same as for [Slice constants](#), and you may also use a constant as a default value. The language mapping guarantees that data members are initialized to their declared default values using a language-specific mechanism.

[Back to Top ^](#)

See Also

- [Structures](#)
- [Constants and Literals](#)
- [Optional Data Members](#)



