Structures

Slice supports structures containing one or more named members of arbitrary type, including user-defined complex types. For example:

Slice

```
struct TimeOfDay {
    short hour;         // 0 - 23
    short minute;       // 0 - 59
    short second;       // 0 - 59
};
```

As in C++, this definition introduces a new type called `TimeOfDay`. Structure definitions form a namespace, so the names of the structure members need to be unique only within their enclosing structure.

Data member definitions using a named type are the only construct that can appear inside a structure. It is impossible to, for example, define a structure inside a structure:

Slice

```
struct TwoPoints {
    struct Point {      // Illegal!
                short x;
                short y;
    };
    Point       coord1;
    Point       coord2;
};
```

This rule applies to Slice in general: type definitions cannot be nested (except for modules, which do support nesting). The reason for this rule is that nested type definitions can be difficult to implement for some target languages and, even if implementable, greatly complicate the scope resolution rules. For a specification language, such as Slice, nested type definitions are unnecessary – you can always write the above definitions as follows (which is stylistically cleaner as well):

Slice

```
struct Point {
    short x;
    short y;
};

struct TwoPoints {      // Legal (and cleaner!)
    Point coord1;
    Point coord2;
};
```

You can specify a default value for a data member that has one of the following types:

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

For example:

Copyright © 2017, ZeroC, Inc.

Slice

```
struct 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.

See Also

- Modules
- Basic Types
- Enumerations
- Sequences
- Dictionaries
- Constants and Literals

Copyright © 2017, ZeroC, Inc.