Modules

Previous Next

On this page:

Modules Reduce Clutter

A common problem in large systems is pollution of the global namespace: over time, as isolated systems are integrated, name clashes become quite likely. Slice provides the `module` construct to alleviate this problem:

Slice

```
module ZeroC
{
    module Client
    {
        // Definitions here...
    }
    module Server
    {
        // Definitions here...
    }
}
```

A module can contain any legal Slice construct, including other module definitions. Using modules to group related definitions together avoids polluting the global namespace and makes accidental name clashes quite unlikely. (You can use a well-known name, such as a company or product name, as the name of the outermost module.)

Back to Top ^

Modules are Mandatory

Slice requires all definitions to be nested inside a module, that is, you cannot define anything other than a module at global scope. For example, the following is illegal:

Slice

```
interface I   // Error: only modules can appear at global scope
{
    // ...
}
```

Definitions at global scope are prohibited because they cause problems with some implementation languages (such as Python, which does not have a true global scope).

> Throughout the Ice manual, you will occasionally see Slice definitions that are not nested inside a module. This is to keep the examples short and free of clutter. Whenever you see such a definition, assume that it is nested in module `M`.

Back to Top ^

Reopening Modules

Modules can be reopened:

Slice

```
module ZeroC
{
    // Definitions here...
}

// Possibly in a different source file:

module ZeroC // OK, reopened module
{
    // More definitions here...
}
```

Reopened modules are useful for larger projects: they allow you to split the contents of a module over several different source files. The advantage of doing this is that, when a developer makes a change to one part of the module, only files dependent on the changed part need be recompiled (instead of having to recompile all files that use the module).

Back to Top ^

Module Mapping

Modules map to a corresponding scoping construct in each programming language. (For example, for C++ and C#, Slice modules map to namespaces whereas, for Java, they map to packages.) This allows you to use an appropriate C++ `using` or Java `import` declaration to avoid excessively long identifiers in your source code.

Back to Top ^

The Ice Module

APIs for the Ice run time, apart from a small number of language-specific calls that cannot be expressed in Ice, are defined in the `Ice` module. In other words, most of the Ice API is actually expressed as Slice definitions. The advantage of doing this is that a single Slice definition is sufficient to define the API for the Ice run time for all supported languages. The respective language mapping rules then determine the exact shape of each Ice API for each implementation language.

We will incrementally explore the contents of the `Ice` module throughout this manual.

Back to Top ^

See Also

- Slice Source Files

Previous

Next