Introduction

The namespaces mechanism in C++ is quite elementary. Since, the history of packaging is known to most readers we will begin with how Z++ views the notion of namespace.

Briefly, here is a list of extension to the way C++ presents a namespace.

- Namespaces can have private/protected sections.
- Namespaces can be derived from one another.
- The definition and implementation of a namespace can be separated.
- The scope of an introduced namespace can be ended.

The last item in the list refers to the "using" statement in C++, which is endless within the scope in which it appears. Note that namespace is a packaging concept, not a type. That is, one cannot create instances of a namespace.

Namespace Sections

A namespace is a sophisticated mechanism for packaging libraries. If all the internals of a library are accessible, the packaging is no more than a bag of items. Generally, there are a few items that a library intends to present to the outside world, and everything else is for its own internal use. The few items to be exported cause much less name clash than exposing the entire namespace.

Z++ uses public section of a namespace for its export mechanism. Items in protected and private sections are not accessible except for derivation purposes. The syntax for designating sections of a namespace as private and public is identical to that of class construct.

Namespace Derivation

Since a namespace does not introduce a type, namespace derivation is not related to polymorphism. Namespace derivation is a convenience for packaging. C++ uses nesting of definitions for adding new items to a namespace. Derivation provides the same benefits along with more control, and better arrangement of code.

A namespace is defined as follows, where "identifier" is name of the namespace being defined.

```
[protected] namespace identifier
        //body
endspace;
```

The specification "protected" is enclosed within meta-symbols indicating that it is optional. A protected namespace can only be used as base in a derivation. That is, it is not open for direct access.

The default access for derivation is public, same as class derivation in Z++. Suppose "Base" is an already defined namespace. Consider the following.

```
namespace Derived : private Base
        //body
endspace;
```

In the above definition, namespace Derived can use Base in the ordinary sense of a namespace. However, since derivation is private, the users of Derived have no access to Base. If Base were declared protected, no other part of the program could use it, at all.

Namespace Implementation

The implementation of function and method bodies can be given in a separate file using the following construct.

```
implementation identifier
        //body
endspace;
```

This allows the construction of static libraries where only the header files including the definitions are needed by users of such libraries.

Namespace Scope

In Z++ one can end a "using" statement with the ending tag "endusing", as follows.

```
using namespace identifier;
        //scope of namespace identifier
endusing namespace identifier;
```