Author: Nikhil Joshi, IBM, nikhilnjoshi@in.ibm.com
Document number: N3349=12-0039
Date: 2012-01-11
Project: Programming Language C++, Evolution Working Group
Reply-to: Nikhil Joshi, IBM, nikhilnjoshi@in.ibm.com
Revision: 1

Ease of using namespaces

1 The Problem

1.1 Problem One

With increase of C++ usage there is more likely chance that there are classes named exactly the same across different namespaces.

When using these seemingly different namespaces in the same piece of code two different classes, named exactly the same, create conflict.

Currently there is only one solution in C++ to resolve this, use of scope resolution operator.

Using scope resolution operator is not an elegant solution; especially when a developer adds a third-party library which has a class named exactly the same as class already defined in legacy code

In above case developer has to go to each line where the class from the legacy code is used and prepend correct namespace name and scope resolution operator. In some cases there might hundreds of such lines that will be required to be changed for just adding a new library into the mix.

1.2 Problem Two

Another problem is unwanted, indirect inclusions because of nested # include directives.

E.g. <Windows.h> implicitly includes headers for <msxml.h> which has class called DomDocument, <header1> includes <Windows.h>. Source file foo.cpp uses <header1> and requires<Windows.h> to be included to compile correctly. Another source file foobity.cpp includes <header1> but does not need functionality/declarations from <Windows.h >, and has declaration of separate class called DomDocument (for example from a xerces Parser). This again creates same conflict.

2 The Proposal

I propose that a new directive “ignore namespace *namespace_name*” to be introduced in C++.

Idea is that the compiler should ignore all symbols from *namespace_one* if it finds “ignore namespace *namespace_one*”

This should solve problems one and two mentioned above, partially.

When developer wants to use a single class from either namespace, without need to use the class with exactly the same name from another namespace, “ignore namespace” directive will solve the problem completely.

E.g. from Problem Two above, developer of foobity.cpp will put line “ignore namespace namespace_for_msxml“, this will result in the compiler ignoring all symbols from indirectly included, not required header file and avoid conflict with the required version of DOMDocument.

When developer needs to mingle these two classes in the same code segment this directive still doesn’t solve the problem completely. For solving such scenario, developer still needs to prepend the each usage of the classes with proper namespace name and scope resolution operator.

3 Interactions and Implementability

3.1 Interactions

This feature should interact with current C++ without affecting exiting code/implementation.

Also, should make it easier for developers to ignore unwanted symbols from namespace.

3.2 Implementability

There are no known or anticipated difficulties in implementing this feature.

4 Conclusion

Though this addition will not completely solve problem of too generic a class names or duplication of symbols in the scope of namespace; it will provide a simpler, easier to implement way of handling the duplication to C++ users.

5 Acknowledgements

I thank Renji Panicker, Arun Kumar for the discussions and resulting suggestions and Michael Wong for his suggestions and help in putting this proposal together.