WG21/N1726=J16/04-0166
Refer to Stroustrup, WG21/N1614=J16/04-0054
2004-10-21

Thomas Plum

Macro Scopes

1. Solve the "#import name-clash" problem (see AvoidingPriorKeywords) by adding an "s" to the keywords, i.e.

```
#imports A, B, C
#exports X, Y, Z
```

2. To define a new "macro scope", use the string "new_scope" in the definition of an inclusion-guard:

```
#ifndef BOOST_WHATEVER_HPP
#define BOOST_WHATEVER_HPP new_scope // = #scope
 ...
#endif // BOOST_WHATEVER_HPP  (serves as #endscope)
```

3. Within the scope of the "new_scope" marker, use the #imports and #exports as originally proposed:

```
#ifndef BOOST_WHATEVER_HPP
#define BOOST_WHATEVER_HPP new_scope

#imports A, B, C
#exports X, Y
#exports Z
 ...
#endif // BOOST_WHATEVER_HPP
```

3. (again) Within the scope of the "new_scope" marker, use the #imports and #exports as originally proposed:

```
#ifndef BOOST_WHATEVER_HPP
#define BOOST_WHATEVER_HPP new_scope

#imports A, B, C
#exports X, Y
#exports Z
 ...
#endif // BOOST_WHATEVER_HPP
```

4. Permit an alternate "#define" representation for the #imports and #exports, such as:

```
#ifndef BOOST_WHATEVER_HPP
#define BOOST_WHATEVER_HPP new_scope

#define BOOST_WHATEVER_HPP_IMPORTS A, B, C
#define BOOST_WHATEVER_HPP_EXPORTS X, Y
#define BOOST_WHATEVER_HPP_EXPORTS Z
 ...
#endif // BOOST_WHATEVER_HPP
```

5. If a preprocessor recognizes and honors the #imports and #exports syntax, then it shall also recognize and honor the "#define" representation. Call this a "new" preprocessor.

6. The "new" preprocessor causes the "local" macro names in each new_scope to be "not visible" outside the new_scope, and the not-imported names from outside do not conflict with the "local" macro names, as per Bjarne's original paper.

7. Obviously, a library vendor targeting multiple platforms will have to use the alternate solution until all their target environments implement #imports and #exports.

8. Along with the "new_scope" marker, today's library vendor can add a "prefix()" marker, listing the name-prefixes used by "local" macro names in today's headers. [Option 8A: If a "prefix()" marker appears, then the inclusion-guard is defined as an allowable prefix, by default.]

```
#ifndef BOOST_WHATEVER_HPP
#define BOOST_WHATEVER_HPP new_scope prefix(BOOST_)

#define BOOST_WHATEVER_HPP_IMPORTS A, B, C
#define BOOST_WHATEVER_HPP_EXPORTS X, Y
#define BOOST_WHATEVER_HPP_EXPORTS Z
 ...
#endif // BOOST_WHATEVER_HPP
```

9. If a header file specifies one or more "prefix" strings, then its names must also meet the "good citizen" property; i.e., each name that is #defined in that header file must be in one of these four sets:

a. Names explicitly listed on "imports" lines.

b. Names explicitly listed on "exports" lines.

c. Names prefixed with a prefix listed in a "prefix()" list.

d. [Option 8A] Names prefixed with the inclusion-guard.

Enforcing this list of four sets is a requirement upon every "new" preprocessor.

Checking HybridSolution against the list of constraints:

By piggy-backing on the inclusion-guard ifdef mechanism we meet the constraints described at InclusionGuards, AvoidingPriorKeywords, InteractionWithIf, NestingBehavior, and LibraryVendors.

The constraint SpecialMacros should be handled the same way in any solution: the language's standard library specifies exactly what names are reserved as macro names to the implementation. The implementation of any solution should honor these reserved names.

See tentative proposal re constraint at InteractionWithUndef

[This paper, and links listed above, refer to “ScopeProposal” on WG14 and WG21 wiki, which was at wiki.plumhall.com/Wg14wg21 as of Oct 21 11am.]