The “scope” extension for the C/C++ preprocessor

Document Number: WG14/N0186
WG21/N1740= J16/04-0180
Date: November 3, 2004
Project: Programming Language C++
Programming Language C
Reference: WG21/N1726=J16/04-0166
ISO/IEC IS 148882:2003(E)
ISO/IEC IS 9899:2002(E)
Reply to: Thomas Plum tplum@plumhall.com
Plum Hall Inc
3 Waihona Box 44610
Kamuela HI 96743 USA

The discussion and brainstorming for the “scope” proposal is open to review and contributions by both committees, at http://wiki.dinkumware.com. Click on “main site for WG14 and WG21”. Login as “wg14” or “wg21” (passwords as given at Redmond, or check with your Head of Delegation), then click on “ScopeProposal”.

As of today, the general outlines of a solution are found at “HybridSolution”, and are quoted below:

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. The syntax and "begin-end" matching of the "new region" marker remains the most contentious issue; see below. For purposes of discussion, let's use an obviously-invalid place-holder:

```
xxx-begin-macro-region
```

```
  ...
xxx-end-macro-region
```

3. Whatever we use for "begin-end" region, the `#imports` and `#exports` can be hidden from "old" preprocessors using the usual feature-test approach. The semantics of `#imports` and `#exports` remain as originally proposed, with some simplifications (see below).

```
#ifdef __std_macro_imports_exports  // or whatever
#imports A, B, C
#exports X, Y
#exports Z
#endif // __std_macro_imports_exports
 ...
```

4. A preprocessor that honors `#imports` and `#exports` is a "new" preprocessor. A "new" preprocessor causes the "local" macro names in each "macro region" to be "not visible" outside the region, and the not-imported names from outside do not conflict with the "local" macro names, as per Bjarne's original paper.

5. Obviously, a library vendor targeting multiple platforms must use pp syntax which can be compiled with today's preprocessors, until all their target environments implement the new pp syntax. The feature-test described above (e.g. `__std_macro_imports_exports`) solves part (or all) of this problem.

6. If a header is wrapped with an inclusion-guard, the macro region(s) should lie strictly within the inclusion-guards:

```
#ifndef GUARD
#define GUARD
#include "other-headers"  // OUTSIDE the macro region
xxx-begin-macro-region
 ...
xxx-end-macro-region
#endif // GUARD
```

7. The work-around for "old" preprocessors could completely rely upon the usual feature-test macro:

```
#ifndef GUARD
#define GUARD

#ifdef __std_macro_imports_exports  // or whatever
#macro-region  // or whatever
#imports A, B, C
#endif // __std_macro_imports_exports
  ...

#ifdef __std_macro_imports_exports  // or whatever
#exports X, Y
#exports Z
#end-macro-region
#endif // __std_macro_imports_exports

#endif // GUARD
```

8. If "helper" macros are needed for exported macros, then require them to be explicitly exported also. (This simplifies the scoped-name requirements considerably.)

```
#ifndef GUARD
#define GUARD

#ifdef __std_macro_imports_exports  // or whatever
#macro-region  // or whatever
#imports A, B, C
#endif // __std_macro_imports_exports

#define Y(i,j,k) ((i)+(j)+(k))
#define Z() 0
#define X() Y(A,B,C) +Z()  // Y and Z are "helpers" of X
```

```
  ...

#ifdef __std_macro_imports_exports  // or whatever
#exports X
#exports Y, Z  // have to explicitly export Y and Z
#end-macro-region
#endif // __std_macro_imports_exports

#endif // GUARD
```

9. Let's try to minimize the burden of "special library macro" names. Obviously `__FILE__`, `__LINE__`, `__STDC*`, `__cplusplus__`, etc (standard feature tests) have to cross the macro-region boundaries without any restrictions. The simplest proposal is

- Macro names in the implementer namespace (beginning with two underscores or underscore+uppercase letter) cross the macro-region boundaries without restrictions;

- All other names must be explitly imported/exported, including names which are library macros in C or C++.

Checking HybridSolution against the list of constraints:

Re constraint SpecialMacros - see above.

Re constraints described at InteractionWithIf, NestingBehavior, and LibraryVendors - see above.

Regarding the InteractionWithIf and NestingBehavior constraints: In the "new" preprocessor, which recognizes the "new" `#macro_region`:

a. when a `#endif` is encountered, it must match a `#if` (and not a `#macro_region`)

b. when a `#end_macro_region` is encountered, it must match a `#macro_region` (and not a `#if`).

(For the purposes of these tests, the feature-test for "macro regions" has to be ignored by the test.)

The AvoidingPriorKeywords constraint is handled by adding "s", making "`#imports`" and "`#exports`".

See tentative proposal re constraint at InteractionWithUndef.
Regarding InclusionGuards: - see above.