NAME

NOTE, _NOTE – annotate source code with info for tools

SYNOPSIS

#include <note.h>
NOTE(*NoteInfo***)**

or

#include <sys/note.h>
_NOTE(*NoteInfo***)**

MT-LEVEL

Safe

DESCRIPTION

These macros are used to embed information for tools in program source. A use of one of these macros is called an "annotation". A tool may define a set of such annotations which can then be used to provide the tool with information that would otherwise be unavailable from the source code.

Annotations should, in general, provide documentation useful to the human reader. If information is of no use to a human trying to understand the code but is necessary for proper operation of a tool, use another mechanism for conveying that information to the tool (one which does not involve adding to the source code), so as not to detract from the readability of the source. The following is an example of an annotation which provides information of use to a tool and to the human reader (in this case, which data are protected by a particular lock, an annotation defined by the static lock analysis tool **lock_lint**).

```
NOTE(MUTEX_PROTECTS_DATA(foo_lock, foo_list Foo))
```

Such annotations do not represent executable code; they are neither statements nor declarations. They should not be followed by a semicolon. If a compiler or tool that analyzes C source does not understand this annotation scheme, then the tool will ignore the annotations. (For such tools, **NOTE(***x***)** expands to nothing.)

Annotations may only be placed at particular places in the source. These places are where the following C constructs would be allowed:

- a top-level declaration (that is, a declaration not within a function or other construct)
- a declaration or statement within a block (including the block which defines a function)
- a member of a **struct** or **union**.

Annotations are not allowed in any other place. For example, the following are illegal:

```
x = y + NOTE(...) z ;
typedef NOTE(...) unsigned int uint ;
```

While **NOTE** and **_NOTE** may be used in the places described above, a particular type of annotation may only be allowed in a subset of those places. For example, a particular annotation may not be allowed inside a **struct** or **union** definition.

NOTE vs _NOTE

Ordinarily, **NOTE** should be used rather than **_NOTE**, since use of **_NOTE** technically makes a program non-portable. However, it may be inconvenient to use **NOTE** for this purpose in existing code if **NOTE** is already heavily used for another purpose. In this case one should use a different macro and write a header file similar to **/usr/include/note.h** which maps that macro to **_NOTE** in the same manner. For example, the following makes **FOO** such a macro:

```
#ifndef _FOO_H
#define _FOO_H
#define FOO _NOTE
#include <sys/note.h>
#endif
```

Public header files which span projects should use **_NOTE** rather than **NOTE**, since **NOTE** may already be used by a program which needs to include such a header file.

NoteInfo **Argument**

The actual *NoteInfo* used in an annotation should be specified by a tool that deals with program source (see the documentation for the tool to determine which annotations, if any, it understands).

NoteInfo must have one of the following forms:

NoteName
NoteName **(***Args***)**

where *NoteName* is simply an identifier which indicates the type of annotation, and *Args* is something defined by the tool that specifies the particular *NoteName.* The general restrictions on *Args* are that it be compatible with an ANSI C tokenizer and that unquoted parentheses be balanced (so that the end of the annotation can be determined without intimate knowledge of any particular annotation).

SEE ALSO

note(4)