Title: Namespaces Without Name Mangling
Author: Marcus Johnson
Company: Mad Scientist
Date: February 16th 2025
Document: n3491
Proposal Category: New Feature
Audience: Compiler authors, Linker authors

Abstract:

Allow symbols (macros, functions, structs, enums) to be referenced by the header they appear in and allow headers to be included via their name.
This proposal could also be called Named Translation Units, that's the core idea.
There is a one to one relationship between namespaces and translation units, there is no nesting namespaces, there is no multiple namespaces per translation unit.

Syntax:

#name StringIO
#include 'PlatformIO'

Rationale:

C++'s namespaces is basically just a giant, multi-translation unit class breaking the translation unit standard and requiring hacks upon hacks to reconcile; my proposal breaks nothing, it simply allows symbols to be referred to by the header they appear in.

Semantics:

At the top of each header should be a #name directive which names the translation unit, (this name is given by the programmer not the compiler)

Any further #name directives are ignored (that way including headers within headers doesn't mess it up)

Identifiers provided to the name directive are the same as any other identifier in C, A-Za-z0-9-_

Namespace identifiers are separated from symbol names with the member access symbol of a period e.g. CommandLineIO.SetName SetName being the name of a function in the CommandLineIO namespace.

The name directive also replaces header guards/pragma once.

Includes also work differently, they're included by name, and the name is surrounded by single quotes to disambiguate named includes from standard path-based includes (both relative path quoted ones and system-path angle bracket denoted includes)

Implementability:
Compilers could simply grep all include paths for each named include,

Example:
#name PlatformIO

```
#include <stdint.h>

size_t MemCopy8(void *Source, void *Destination, size_t NumElements);

size_t MemCopy16(void *Source, void *Destination, size_t NumElements);

#define MemCopy(Source, Destination, NumElements) _Generic((Source), uint8_t:MemCopy8, uint16_t:MemCopy16)(Source, Destination, NumElements)
```

#name StringIO

```
#include ‘PlatformIO’
#include <string.h>
#include <stdlib.h>

typedef struct UTF8 {
   size_t NumCodeUnits;
   char8_t *String;
} UTF8;

UTF8 UTF8_InitializeFromLiteral(char8_t *Literal) {
   UTF8 String = {};
   size_t LiteralSize = strlen(Literal);
   String.String = calloc(LiteralSize);
   size_t NumElementsCopied = PlatformIO.MemCopy(Literal, String.String, LiteralSize);
   String.NumCodeUnits = NumElementsCopied;
   return String;
}
```