parc®
A Xerox Company

Computing Science Laboratory
Notes on Software Design

Naming Conventions in C Programs

Glenn Scott[1]

Abstract
This is a design guide for the management of modularity and coupling in the C programming language.

Keywords
Design Guide — C Language — Software Quality

[1] *Computing Science Laboratory, Palo Alto Research Center*

Contents

1. Introduction

The C programming language offers no provisions for managing the names of functions, structures, type definitions, and enumerations. Each of these kinds of things exist in their own flat-namespace shared with every other named element of their kind. For example, within a C program there can only be one function named `write`, one structure named `result`, and so forth.

Software architects and implementors must manage these namespaces manually (mentally). As the source code for a software system grows, the cognitive burden also grows, leading to a increasingly difficult system to learn and use. Typically, this culminates in a system that is understood and maintained by only a small group, finding themselves struggling against any changes, and is rarely used to its fullest extent.

This memo is a description of an obvious technique to manage namespaces in C programs and elaborates on some useful conventions for that technique to increase usability.

2. Name Spaces in C

The namespace scheme is simply a linearized form of a path through a tree, where the path is a concatenation of the names of the nodes of the tree terminated by the name of the leaf. The leaf and its prefix are separated by a single _ (underbar) character.

Applying the convention starts by choosing the root name for the namespace. For example, the namespace for the PARC Library is simply `PARC` and the namespace for the LongBow unit-test system is `LongBow`. For the best long-term results, choose a name that will be sufficiently unique and relevant to its purpose. The name of the root namespace appears in both its upper- and lower-case forms (`PARC` and `parc`, `LongBow` and `longBow`) to give lexical and visual differentiation and semantic meaning in specific contexts which are described below.

$\boxed{\text{namespace}}\_\boxed{\text{name}}.c$

3. Name Spaces and Files

All C source and header files are named with the lower-case form of the name as the prefix `parc` followed by a single _ character separating the name of the namespace from the the name of the module. The Buffer implementation in the PARC Library has the name `parc_Buffer` and the corresponding source and header files are named `parc_Buffer.c` and `parc_Buffer.h`.

Sometimes sub-modules are a necessary design element and naming sub-modules follow the same pattern. For example, `longBow_Report.c` and `longBow_Report.h`, implements a facade to a set of functions implementing a reporting mechanism for LongBow. These functions have different implementations depending upon the output format. As a consequence, the submodules implementing these functions use the namespace ‘longBowReport‘ followed by the _ character followed by the component name, where a component is a function, a constant (both enumerations and preprocessor defines), and types.

4. Name Spaces and Functions

Functions are declared and defined in corresponding files, and following the same pattern, the function is named with its namespace prefix followed by the _ character and suffixed by the function?s name.

For example, the function that returns the current capacity of a PARC Library Buffer is declared and defined named, `parcBuffer_Capacity` Where the namespace prefix is the composite of ‘parc‘ and ‘Buffer‘ with the _ character again separating the name ‘Capacity‘.

5. Name Spaces and Constants

Named constant appear as either C preprocessor defines, or as C ‘enum‘ types.

C modules implement various kinds of functionality ranging from defining constants, utility functions, and composite types and combinations of these. The names for each of these continue the naming scheme of separating the namespace from the component name (function, constant, and types).

5.1 Naming Functions

5.2 Naming Types