

## Example function

### Pre-function comment

- Before a function put a comment, starting with `/**`
- Explain the purpose of the function.
- List the parameters and what they contain. If they are updated in the function, say so, and say what they are updated with.
- If the function returns something, say what it is.

### Function

- Function name is `lower_case_with_underscores`
- No whitespace between function name and parameter brackets `function(...)`
- Parameter ordering is: input, input/output, output.
- If line exceeds 80 characters, split the line and indent remainder with two tabs. Always split parameter lists after a comma.
- Open function block curly brace on new line.
- Contents are indented with one tab.

### General

- Always use `/* ... */` comment style.
- `for`, `while`, `if` etc are not functions, so there is a space between the keyword and the condition brackets.
 

```
if (condition1) {
    /* code */
} else if (condition2) {
    /* code */
} else {
    /* code */
}
```
- Non-function curly brackets are placed on the same line as the keyword they relate to. They are separated from other content on the line by a space.
- One tab indentation.
- Two tabs for wrapped lines.

```
/**
 * Solve the width constraint as given in CSS 2.1 section 10.3.3.
 * \param available_width Max width available in pixels
 * \param width Current box width
 * \param margin[4] Current box margins. Updated with new box
 *                 left / right margins
 * \param padding[4] Current box paddings. Updated with new box
 *                  left / right paddings
 * \param border[4] Current box border widths. Updated with new
 *                 box left / right border widths
 * \return New box width
 */
int layout_solve_width(int available_width, int width,
                      int margin[4], int padding[4], int border[4])
{
    if (width == AUTO) {
        /* any other 'auto' became 0 */
        if (margin[LEFT] == AUTO)
            margin[LEFT] = 0;
        if (margin[RIGHT] == AUTO)
            margin[RIGHT] = 0;
        width = available_width -
            (margin[LEFT] + border[LEFT] + padding[LEFT] +
             padding[RIGHT] + border[RIGHT] + margin[RIGHT]);
    } else if (margin[LEFT] == AUTO && margin[RIGHT] == AUTO) {
        /* make the margins equal, centering the element */
        margin[LEFT] = margin[RIGHT] = (available_width -
            (border[LEFT] + padding[LEFT] + width +
             padding[RIGHT] + border[RIGHT])) / 2;
        if (margin[LEFT] < 0) {
            margin[RIGHT] += margin[LEFT];
            margin[LEFT] = 0;
        }
    } else if (margin[LEFT] == AUTO) {
        margin[LEFT] = available_width -
            (border[LEFT] + padding[LEFT] + width +
             padding[RIGHT] + border[RIGHT] + margin[RIGHT]);
    } else {
        /* margin-right auto or "over-constrained" */
        margin[RIGHT] = available_width -
            (margin[LEFT] + border[LEFT] + padding[LEFT] +
             width + padding[RIGHT] + border[RIGHT]);
    }

    return width;
}
```

From render/layout.c

## Other details

### General

- NetSurf is written in C89 with C99 struct initialisers and `<stdbool.h>`. We wrap at 80 characters and our tab width is 8 characters.
- Use `true` and `false` for booleans, unless interacting with library code which uses `TRUE` and `FALSE`.
- Put an empty line at the end of every source file.

### Modules

- Source code is organised into modules. A module is normally a `.c` and `.h` file. For example, the html module is `html.c` and `html.h`.
- All functions (including static functions) in a module should start `<module>_`.
 

```
html_create()
html_process_data()
```

 This makes functions easy to find and unique through the source, which is helpful for backtraces and documentation.
- Global variables should also start with `<module>_`.
- Functions go in a logical order, for example any init function first, then functions it calls, then the next externally available function, functions it calls.
- Static functions should all be declared at the top.

### Specifics

- Format `switch` statements as follows:
 

```
switch (foo) {
case 1:
    /* code */
    break;
case 2:
    /* code */
    /* fall through */
case 3:
{
    int local;
    /* code */
}
break;
}
```