# Placeholders 3/place

*Purpose*

To manage placeholder variables.

---

3/place.§1-7 Initial values

---

*Definitions*

**¶1.** Placeholders are markers such as "[AUTHOR]", found in the template files for making web pages. ("AUTHOR" would be the name of this one; the use of capital letters is customary but not required.) Most of these can be set to arbitrary texts by use of the `placeholder` command in the blurb file, but a few are "reserved" by `cblorb`:

```
define SOURCE_RPL 1
define SOURCENOTES_RPL 2
define SOURCELINKS_RPL 3
define COVER_RPL 4
define DOWNLOAD_RPL 5
define AUXILIARY_RPL 6
define PAGENUMBER_RPL 7
define PAGEEXTENT_RPL 8
```

```
typedef struct placeholder {
    char pl_name[MAX_VAR_NAME_LENGTH];
    char pl_contents[MAX_FILENAME_LENGTH];                    current value
    int reservation;                          one of the *_RPL values above, or 0 for unreserved
    int locked;                          currently being expanded: locked to prevent mise-en-abyme
    MEMORY_MANAGEMENT
} placeholder;
```

The structure placeholder is private to this section.

---

§**1. Initial values.** The `BLURB` refers here to back-cover-style text, and not to the "blurb" file which we are acting on.

```
void initialise_placeholders(void) {
    set_placeholder_to("SOURCE", "", SOURCE_RPL);
    set_placeholder_to("SOURCENOTES", "", SOURCENOTES_RPL);
    set_placeholder_to("SOURCELINKS", "", SOURCELINKS_RPL);
    set_placeholder_to("COVER", "", COVER_RPL);
    set_placeholder_to("DOWNLOAD", "", DOWNLOAD_RPL);
    set_placeholder_to("AUXILIARY", "", AUXILIARY_RPL);
    set_placeholder_to("PAGENUMBER", "", PAGENUMBER_RPL);
    set_placeholder_to("PAGEEXTENT", "", PAGEEXTENT_RPL);
    set_placeholder_to("CBLORBERRORS", "", 0);
    set_placeholder_to("INBROWSERPLAY", "", 0);
    set_placeholder_to("BLURB", "", 0);
    set_placeholder_to("TEMPLATE", "Standard", 0);
    set_placeholder_to("GENERATOR", VERSION, 0);
    set_placeholder_to("BASE64_TOP", "", 0);
    set_placeholder_to("BASE64_TAIL", "", 0);
```

```
    set_placeholder_to("JAVASCRIPTPRELUDE", JAVASCRIPT_PRELUDE, 0);
    set_placeholder_to("FONTTAG", FONT_TAG, 0);
    initialise_time_variables();
}
```

The function initialise_placeholders is called from 1/main.

§**2.** We don't need any very efficient system for parsing these names, as there are typically fewer than 20 placeholders at a time.

```
placeholder *find_placeholder(char *name) {
    placeholder *wv;
    LOOP_OVER(wv, placeholder)
        if (strcmp(wv->pl_name, name) == 0)
            return wv;
    return NULL;
}
char *read_placeholder(char *name) {
    placeholder *wv = find_placeholder(name);
    if (wv) return wv->pl_contents;
    return NULL;
}
```

The function find_placeholder is.
The function read_placeholder is called from 1/main, 1/blurb, 3/rel, 3/links and 3/web.

§**3.** There are no "types" of these placeholders. When they hold numbers, it's only as the text of a number written out in decimal, so:

```
void set_placeholder_to_number(char *var, int v) {
    char temp_digits[64];
    sprintf(temp_digits, "%d", v);
    set_placeholder_to(var, temp_digits, 0);
}
```

The function set_placeholder_to_number is called from 1/main and 1/blurb.

§**4.** And here we set a given placeholder to a given text value. If it doesn't already exist, it will be created. A reserved placeholder can then never again be set, and since it will have been set at creation time (above), it follows that a reserved placeholder cannot be set with the `placeholder` command of a blurb file.

```
void set_placeholder_to(char *var, char *text, int reservation) {
    set_placeholder_to_inner(var, text, reservation, FALSE);
}
void append_to_placeholder(char *var, char *text) {
    set_placeholder_to_inner(var, text, 0, TRUE);
}
```

The function set_placeholder_to is called from 1/main, 1/blurb and 3/rel.
The function append_to_placeholder is called from 1/text, 1/blurb and 3/rel.

§**5.** Where:

```
void set_placeholder_to_inner(char *var, char *text, int reservation, int extend) {
    if (strlen(var) >= MAX_VAR_NAME_LENGTH-1) { error("variable name too long"); return; }
    if (trace_mode) printf("! [%s] <-- \"%s\"\n", var, (text)?text:"");
    placeholder *wv = find_placeholder(var);
    if ((wv) && (reservation > 0)) { error("tried to set reserved variable"); return; }
    if (wv == NULL) {
        wv = CREATE(placeholder);
        if (trace_mode) printf("! Creating [%s]\n", var);
        strcpy(wv->pl_name, var);
        (wv->pl_contents)[0] = 0;
        wv->reservation = reservation;
    }
    int L = strlen(text) + 1;
    if (extend) L += strlen(wv->pl_contents);
    if (L >= MAX_FILENAME_LENGTH) { error("placeholder text too long"); return; }
    if (extend) strcat(wv->pl_contents, text);
    else strcpy(wv->pl_contents, text);
}
```

The function set_placeholder_to_inner is.

§**6.** And that just leaves writing the output of these placeholders. The scenario here is that we're copying HTML over to make a new web page, but we've hit text in the template like "[AUTHOR]". We output the value of this placeholder instead of that literal text. The reserved placeholders output as special gadgets instead of any fixed text, so those all call suitable routines elsewhere in `cblorb`.

If the placeholder name isn't known to us, we print the text back, so that the original material will be unchanged. (This is in case the original contains uses of square brackets which aren't for placeholding.)

```
int escape_quotes_mode = 0;
void copy_placeholder_to(char *var, FILE *COPYTO) {
    int multiparagraph_mode = FALSE, eqm = escape_quotes_mode;
    if (var[0] == '*') { var++; escape_quotes_mode = 1; }
    if (var[0] == '*') { var++; escape_quotes_mode = 2; }
    if (strcmp(var, "BLURB") == 0) multiparagraph_mode = TRUE;
    placeholder *wv = find_placeholder(var);
    if ((wv == NULL) || (wv->locked)) {
        fprintf(COPYTO, "[%s]", var);
    } else {
        wv->locked = TRUE;
        if (multiparagraph_mode) fprintf(COPYTO, "<p>");
        switch (wv->reservation) {
            case 0: ⟨Copy an ordinary unreserved placeholder 7⟩; break;
            case SOURCE_RPL: expand_SOURCE_or_SOURCENOTES_variable(COPYTO, FALSE); break;
            case SOURCENOTES_RPL: expand_SOURCE_or_SOURCENOTES_variable(COPYTO, TRUE); break;
            case SOURCELINKS_RPL: expand_SOURCELINKS_variable(COPYTO); break;
            case COVER_RPL: expand_COVER_variable(COPYTO); break;
            case DOWNLOAD_RPL: expand_DOWNLOAD_variable(COPYTO); break;
            case AUXILIARY_RPL: expand_AUXILIARY_variable(COPYTO); break;
            case PAGENUMBER_RPL: expand_PAGENUMBER_variable(COPYTO); break;
            case PAGEEXTENT_RPL: expand_PAGEEXTENT_variable(COPYTO); break;
        }
```

```
        if (multiparagraph_mode) fprintf(COPYTO, "</p>");
        wv->locked = FALSE;
        escape_quotes_mode = eqm;
    }
}
```

The function copy_placeholder_to is called from 3/sol and 3/web.

§**7.** Note that the [BLURB] placeholder – which holds the story description, and is like a back cover blurb for a book; the name is not related to the release instructions format – may consist of multiple paragraphs. If so, then they will be divided by `<br/>`, since that's the XML convention. But we want to translate those breaks to `</p><p>`, closing an old paragraph and opening a new one, because that will make the blurb text much easier to style with a CSS file. It follows that [BLURB] should always appear in templates within an HTML paragraph.

⟨Copy an ordinary unreserved placeholder 7⟩ ≡

```
    int i; char *p = wv->pl_contents;
    for (i=0; p[i]; i++) {
        if ((p[i] == '<') && (p[i+1] == 'b') && (p[i+2] == 'r') &&
            (p[i+3] == '/') && (p[i+4] == '>') && (multiparagraph_mode)) {
            fprintf(COPYTO, "</p><p>"); i += 4; continue;
        }
        if (p[i] == '[') {
            char inner_name[MAX_VAR_NAME_LENGTH+1];
            int j = i+1, k = 0, expanded = FALSE; inner_name[0] = 0;
            for (; p[j]; j++) {
                if ((p[j] == '[') || (p[j] == ' ')) break;
                if (p[j] == ']') {
                    i = j;
                    copy_placeholder_to(inner_name, COPYTO);
                    expanded = TRUE;
                    break;
                }
                inner_name[k++] = p[j]; inner_name[k] = 0;
                if (k >= MAX_VAR_NAME_LENGTH) break;
            }
            if (expanded) continue;
        }
        if (((p[i] == '\x0a') || (p[i] == '\x0d') || (p[i] == '\x7f')) &&
            (multiparagraph_mode)) {
            fprintf(COPYTO, "<p>"); continue;
        }
        if ((escape_quotes_mode == 1) && (p[i] == '\'')) fprintf(COPYTO, "&#39;");
        else if ((escape_quotes_mode == 2) && (p[i] == '\'')) fprintf(COPYTO, "%%2527");
        else fprintf(COPYTO, "%c", p[i]);
    }
```

This code is used in §6.