Links and Auxiliary Files 3/links

Purpose

To manage links to auxiliary files, and placeholder variables.

3/links.§1 Registration; §2-3 Linking; §4-5 Links; §6 Cover image; §7 Releasing

---

Definitions

¶1. Auxiliary files are for items bundled up with the release but which are deliberately made accessible for the eventual player: things such as maps or manuals. `cblorb` needs to know about these only when releasing a website; they are also recorded in an iFiction record, but `cblorb` does not create that (`ni` does).

```
typedef struct auxiliary_file {
    char relative_URL[MAX_FILENAME_LENGTH];
    char full_filename[MAX_FILENAME_LENGTH];
    char aux_leafname[MAX_FILENAME_LENGTH];
    char description[MAX_FILENAME_LENGTH];
    char format[MAX_EXTENSION_LENGTH];                    e.g., "jpg", "pdf"
    MEMORY_MANAGEMENT
} auxiliary_file;
```

The structure auxiliary_file is private to this section.

---

§**1. Registration.** The format text is set to a lower-case version of the filename extension, and the URL to the filename itself; except when there is no extension, so that the auxiliary resource is a mini-website in a subfolder of the release website. In that case the format is `link` and the URL is to the index file in the subfolder.

```
void create_auxiliary_file(char *filename, char *description) {
    auxiliary_file *aux = CREATE(auxiliary_file);
    strcpy(aux->description, description);
    strcpy(aux->full_filename, filename);
    char *ext = get_filename_extension(filename);
    char *leaf = get_filename_leafname(filename);
    if (ext[0] == '.') {
        strcpy(aux->relative_URL, filename);
        if (strlen(ext + 1) >= MAX_EXTENSION_LENGTH - 1) {
            error("auxiliary file has overlong extension"); return;
        }
        strcpy(aux->format, ext + 1);
        int k; for (k=0; aux->format[k]; k++) aux->format[k] = tolower(aux->format[k]);
    } else {
        strcpy(aux->format, "link");
        sprintf(aux->relative_URL, "%s%cindex.html", filename, SEP_CHAR);
    }
    strcpy(aux->aux_leafname, leaf);
    printf("! Auxiliary file: <%s> = <%s>\n", filename, description);
}
```

The function create_auxiliary_file is called from 1/blurb.

§2. **Linking.** The list of links to auxiliary resources is written using `<li>...</li>` list entry tags, for convenience of CSS styling.

```
void expand_AUXILIARY_variable(FILE *COPYTO) {
    auxiliary_file *aux;
    LOOP_OVER(aux, auxiliary_file) {
        fprintf(COPYTO, "<li>");
        download_link(COPYTO,
            aux->description, aux->full_filename, aux->aux_leafname, aux->format);
        fprintf(COPYTO, "</li>");
    }
    add_links_to_requested_resources(COPYTO);
}
```

The function expand_AUXILIARY_variable is.

§3. On some of the pages produced by `cblorb` the story file itself looks like another auxiliary resource, but it's produced thus:

```
void expand_DOWNLOAD_variable(FILE *COPYTO) {
    char target_pathname[MAX_FILENAME_LENGTH];                    eventual pathname of Blorb file written
    sprintf(target_pathname, "%s%c%s", release_folder, SEP_CHAR, read_placeholder("STORYFILE"));
    download_link(COPYTO, "Story File", target_pathname, read_placeholder("STORYFILE"), "Blorb");
}
```

The function expand_DOWNLOAD_variable is.

§4. **Links.** This routine, then, handles either kind of link.

```
void download_link(FILE *COPYTO, char *desc, char *filename, char *relative_url, char *form) {
    int size_up = TRUE;
    if (strcmp(form, "link") == 0) size_up = FALSE;
    fprintf(COPYTO, "<a href=\"%s\">%s</a> ", relative_url, desc);
    open_style(COPYTO, "filetype");
    fprintf(COPYTO, "(%s", form);
    if (size_up) {
        long int size = -1L;
        if (strcmp(desc, "Story File") == 0) size = (long int) blorb_file_size;
        else size = file_size(filename);
        if (size != -1L) ⟨Write a description of the rough file size 5⟩
    }
    fprintf(COPYTO, ")");
    close_style(COPYTO, "filetype");
}
```

The function download_link is called from 3/rel.

§**5.** We round down to the nearest KB, MB, GB, TB or byte, as appropriate. Although this will describe a 1-byte auxiliary file as "1 bytes", the contingency seems remote.

⟨Write a description of the rough file size 5⟩ ≡

```
char *units = " bytes";
long int remainder = 0;
if (size > 1024L) { remainder = size % 1024L; size /= 1024L; units = "KB"; }
if (size > 1024L) { remainder = size % 1024L; size /= 1024L; units = "MB"; }
if (size > 1024L) { remainder = size % 1024L; size /= 1024L; units = "GB"; }
if (size > 1024L) { remainder = size % 1024L; size /= 1024L; units = "TB"; }
fprintf(COPYTO, ", %d", (int) size);
if ((size < 100L) && (remainder >= 103L)) fprintf(COPYTO, ".%d", (int) (remainder/103L));
fprintf(COPYTO, "%s", units);
```

This code is used in §4.

§**6. Cover image.** Note that if the large cover image is a PNG, so is the small (thumbnail) version, and vice versa – supplying "Cover.jpg" and "Small Cover.png" will not work.

```
void expand_COVER_variable(FILE *COPYTO) {
    if (cover_exists) {
        char *format = "png"; if (cover_is_in_JPEG_format) format = "jpg";
        fprintf(COPYTO, "<a href=\"Cover.%s\"><img src=\"Small Cover.%s\" border=\"1\" /></a>",
            format, format);
    }
}
```

The function expand_COVER_variable is.

§**7. Releasing.** When we generate a website, we need to copy the auxiliary files into it (though not mini-websites: the user will have to do that).

```
void request_copy_of_auxiliaries(void) {
    auxiliary_file *aux;
    LOOP_OVER(aux, auxiliary_file)
        if (strcmp(aux->format, "link") != 0) {
            if (trace_mode)
                printf("! COPY <%s> as <%s>\n", aux->full_filename, aux->aux_leafname);
            request_copy(aux->full_filename, aux->aux_leafname);
        }
}
```

The function request_copy_of_auxiliaries is called from 3/rel.