

```

/*****
 *
 * NeWS HyperTIES formatter
 * Don Hopkins
 *
 *****/

#include <stdio.h>
#include <sys/param.h>
#include "fmt.h"
#include "entry.h"
#include "master-index.h"

/*****/

#define MAX_STR_LEN 512

/*****/

/*
 * Font metrics cache
 *
 * This implements a font metrics cache in the client.
 * When a font of some point size not in the cache is needed, NeWS is
 * queried for the height, ascent, and descent of the font, and the
 * widths of all 256 characters.
 */

/*
 * The metrics are just in a singly linked list right now. This should
 * eventually be changed to use a tree so looking up fonts is faster.
 */

struct font_metrics {
    char *name;
    int size;
    int ascent;
    int descent;
    int height;
    int *widths;
    struct font_metrics *next;
};

struct font_metrics *cached_metrics = NULL;

/*
 * Query the NeWS server for the metrics of a font of a given name and point.
 * Return NULL if it was not found.
 * Allocate a font_metrics struct, fill it in and load the metrics into it,
 * add it to the cache, and return a pointer to it.
 */
struct font_metrics *
get_metrics(name, size)
    char *name;
    int size;
{
    int i, c;
    struct font_metrics *metrics;

```

```

do_find_font(name, size);
ps_flush_PostScript();

while (1) {
    if (psio_error(PostScriptInput)) {
        fprintf(stderr, "Error on input!\n");
        bomb(1);
    } else if (found_font()) {
        metrics = (struct font_metrics *)calloc(1, sizeof(struct font_metrics));
        if (metrics == NULL ||
            (metrics->widths = (int *)calloc(256, sizeof(int))) == NULL) {
            fprintf(stderr, "Malloc failed!\n");
            bomb(1);
        }
        metrics->name = (char *)malloc(strlen(name)+1);
        strcpy(metrics->name, name);
        metrics->size = size;
        do_get_metrics(&metrics->height, &metrics->ascent, &metrics->descent);
        /* Look out! Here come 256 integers! */
        for (i=0; i<256; i++) {
            pscanf(PostScriptInput, "d", &metrics->widths[i]);
        }
        metrics->next = cached_metrics;
        cached_metrics = metrics;
        return(metrics);
    } else if (no_font()) {
        fprintf(stderr, "Font %s %d not found!\n", name, size);
        return(NULL);
    } else if (psio_eof(PostScript)) {
        fprintf(stderr, "Lost server!\n");
        bomb(1);
    } else {
        c = psio_getc(PostScriptInput);
        printf("Bizarre input '%c' (0x%X)!\n", c, c);
        if (c < 0)
            bomb(1);
    }
}
}

/*
 * Look in the cache for the metrics of a font of a given name and size,
 * and return that if found, otherwise, query the server for the font,
 * and return the result of that.
 */
struct font_metrics *
find_metrics(name, size)
    char *name;
    int size;
{
    register struct font_metrics *cache = cached_metrics;

    while (cache != NULL) {
        if (cache->size == size)
            if (strcmp(cache->name, name) == 0)
                return(cache);
        cache = cache->next;
    }
}

```

```

    }
    return(get_metrics(name, size));
}

/*
 * Determine the length of a string, using a given set of font metrics.
 */
int
string_width(metrics, string)
    struct font_metrics *metrics;
    char *string;
{
    int width;

    for (width=0; *string != '\0'; string++)
        width += metrics->widths[*string];
    return(width);
}

/*****

/*
 * Entry instantiation stuff.
 */

MASTER_INDEX *Master;

/*
 * Unique id for each stamp-pad object.
 * The handle we use to refer to an object in the NeWS server.
 */
int instance_id_count = 0;

char object_class[MAX_STR_LEN];

/*
 * Pump the body of an object into the NeWS server, which
 * executes it, pushing the instantiation arguments onto the
 * stack. Return the name of the object's class, to which a
 * /new message will be sent.
 */

char *send_entry_body(ent)
    ENTRY *ent;
{
    int count, len;
    char buf[MAX_STR_LEN];
    FILE *f;

    if ((f = fopen(FileOfEntry(ent), "r")) == NULL) {
        perror("fopen entry file");
        fprintf(stderr, "Can't open file %s of entry %s!\n",
            FileOfEntry(ent), KeyOfEntry(ent));
        bomb(1);
    }

    fseek(f, OffsetOfEntry(ent), 0);

```

```

count = LengthOfEntry(ent);

fgets(object_class, MAX_STR_LEN, f);
len = strlen(object_class);
object_class[len-1] = '\0';
if (count != -1)
    count -= len;

fgets(buf, MAX_STR_LEN, f);
len = strlen(buf);
if (count != -1)
    count -= len;

while (!feof(f) && (count == -1 || count > 0)) {
    len = ((count > MAX_STR_LEN) || (count == -1) ? MAX_STR_LEN : count);
    len = fread(buf, 1, len, f);
    if (fwrite(buf, 1, len, PostScript) != len) {
        perror("writing entry body");
        fprintf(stderr, "Error writing entry body to PostScript file!\n");
    }
    if (count != -1)
        count -= len;
}

fclose(f);

return(object_class);
}

/*
 * Make sure an stamp-pad picture object is instantiated in memory (as an
 * InstanceHeader) and in the server (as an Stamp in StampDict). If
 * it's not, tell the server where in the file system the object's
 * coming from, pump in the object's body, and send a new message to
 * the object's class, binding it to a unique instance id in
 * StampDict. The instance ID is be used as a handle to the object in
 * the server. Query its size, and cache that away in the InstanceHeader.
 * An picture is a stamp-pad that can put instructions into the display list.
 */
define_picture(ent)
    ENTRY *ent;
{
    INSTANCE_HEADER *new;
    char *class;

    if (InstanceOfEntry(ent) == NULL) {
        new = CreateInstanceHeader(ent, instance_id_count++, 0, 0);
        send_entry_pos(ent);
        class = send_entry_body(ent);
        do_def_picture(class, KeyOfEntry(ent), InstanceId(new));
        do_measure_stamp(InstanceId(new),
                        &InstanceWidth(new), &InstanceHeight(new));
    }
}

/*
 * Make sure a target stamp-pad object is instantiated in memory (as
 * an InstanceHeader) and in the server (as a TargetStamp in

```

```

* StampDict). Same as define_picture, except that a TargetStamp is a
* stamp pad that can put a target item down on the current page,
* enter it into the TargetDict, and put a referece to it in the
* page's Targets dictionary, so the page's event manager can look out
* for its start interests. TargetStamps can also put instructions into
* the display list. Each target that a TargetStamp stamps down gets a
* unique ID, consisting of its name (that the author refered to it
* by), a dot, and a serial number (the count of how many of targets
* of this name have been stamped down). This ID is associated with
* the target object in TargetDict. Each entry in PageDict has a
* Targets field, that is a dictionary associating the target ID of
* each target on that page with its reference.
*/
define_target(ent)
    ENTRY *ent;
{
    INSTANCE_HEADER *new;
    char *class;

    if (InstanceOfEntry(ent) == NULL) {
        new = CreateInstanceHeader(ent, instance_id_count++, 0, 0);
        send_entry_pos(ent);
        class = send_entry_body(ent);
        do_def_target(class, KeyOfEntry(ent), InstanceId(new));
        do_measure_stamp(InstanceId(new),
                        &InstanceWidth(new), &InstanceHeight(new));
    }
}

send_entry_pos(ent)
    ENTRY *ent;
{
    char *slash;
    char dir[MAX_STR_LEN], file[MAX_STR_LEN];
    char *rindex();

    strcpy(dir, FileOfEntry(ent));
    if ((slash = rindex(dir, '/')) == 0) {
        strcpy(file, dir);
        dir[0] = '\0';
    } else {
        strcpy(file, slash+1);
        slash[1] = '\0';
    }
    do_def_file_pos(dir, file, OffsetOfEntry(ent), LengthOfEntry(ent));
}

/*
* Try to find a document of a given name, returning a counted string in buf,
* containing the file name, or a string of zero length, if the name was not
* known.
*/
ENTRY *lookup_document(name, buf)
    char *name;
    char *buf;
{
    ENTRY *ent = FindInIndex(Master->documents, name);

```

```

    if (ent == NULL) {
        fprintf(stderr, "Can't find document name \"%s\" in master index!\n",
            name);
        fflush(stderr);
    }
    counted_entry_name(ent, buf);
    return(ent);
}

```

```

counted_entry_name(ent, buf)
    ENTRY *ent;
    char *buf;
{
    if (ent == NULL) {
        buf[0] = buf[1] = 0;
    } else {
        strcpy(buf+1, FileOfEntry(ent));
        buf[0] = strlen(FileOfEntry(ent));
    }
}

```

```

/*****

```

```

/*
 * Formatting stuff
 *
 * HyperTIES formatter library
 */

```

```

int win_x = 512;
int win_y = 0;
int win_width = 640;
int win_height = 900;
int top_margin = 10;
int bottom_margin = 10;
int left_margin = 10;
int right_margin = 10;
int cur_x, cur_y;
int image_x, image_y;
int image_width, image_height;
int in_line = 0;
int line_x, line_y;
char *source_name = "Untitled";
int source_pos, source_len;
int str_len = 0;
char str[MAX_STR_LEN];
int str_x, str_y;
int str_width;
int in_string = 0;
int page_width, page_height;
int top, bottom, left, right;
int on_page = 0;
char *font_name;
int font_size;
struct font_metrics *font = NULL;
char *default_font_name;
int default_font_size;

```

```

struct font_metrics *default_font = NULL;
char *button_font_name;
int button_font_size;
struct font_metrics *button_font = NULL;
char *definition_font_name;
int definition_font_size;
struct font_metrics *definition_font = NULL;
char *controls_font_name;
int controls_font_size;
struct font_metrics *controls_font = NULL;
int line_height = 0;
int default_line_height = 0;
int line_space = 4;
int para_indent = 4;
int space_width;
int pile_id_count = 0;
char *button_name = "default-button";
char *page_bg_name = NULL;
char *page_bg_ref = NULL;
char *definition_bg_name = NULL;
char *definition_bg_ref = NULL;
char *controls_bg_name = NULL;
char *controls_bg_ref = NULL;
char *full_entry_button_name = NULL;
char *full_entry_button_ref = NULL;
int recording = 0;

```

```

init_margins()
{
    left = left_margin;
    right = page_width - right_margin;
    bottom = bottom_margin;
    top = page_height - top_margin;
}

```

```

move_cur(x, y)
int x, y;
{
    end_line();
    cur_x = x;
    cur_y = y;
    start_line();
}

```

```

rmove_cur(x, y)
int x, y;
{
    move_cur(cur_x + x, cur_y + y);
}

```

```

overstrike()
{
    move_cur(image_x, image_y + image_height);
}

```

```

home()
{
    move_cur(left, top);
}

```

```

    /* first target will cover the whole window */
    place_image(0, 0, page_width, page_height);
    line_height = 0;
}

int new_pile(class, x, y, width, height)
    char *class;
    int x, y, width, height;
{
    do_make_pile(class, pile_id_count, x, y, width, height);
    return(pile_id_count++);
}

start_page()
{
    if (on_page) {
        end_page();
    }

    actually_start_page();
    record_start_page();
    do_get_page_size(&page_width, &page_height);
    init_margins();
    if (page_bg_name != NULL && page_bg_ref != NULL) {
        stamp_background_target(page_bg_name, page_bg_ref);
    }
    home();
    if (font)
        setup_font();
}

record_start_page()
{
    if (recording) {
        printf(".SP\n");
    }
}

actually_start_page()
{
    on_page = 1;
    do_start_page();
}

end_page()
{
    if (on_page) {
        end_line();
        actually_end_page();
        record_end_page();
    }
}

record_end_page()
{
    if (recording) {
        printf(".EP\n");
    } else {

```



```

        printf("p\n");
    }
}

actually_end_page()
{
    on_page = 0;
    do_end_page();
}

start_controls()
{
    if (on_page)
        end_page();
    on_page = 1;

    zap_pages();
    actually_start_page();
    record_start_page();
    do_get_page_size(&page_width, &page_height);
    init_margins();
    if (controls_bg_name != NULL && controls_bg_ref != NULL) {
        stamp_background_target(controls_bg_name, controls_bg_ref);
    }
    bottom = -999999;
    home();
    if (font)
        setup_font();
}

start_definition()
{
    if (on_page)
        end_page();
    on_page = 1;

    zap_pages();
    actually_start_page();
    record_start_page();
    do_get_page_size(&page_width, &page_height);
    init_margins();
    if (definition_bg_name != NULL && definition_bg_ref != NULL) {
        stamp_background_target(definition_bg_name, definition_bg_ref);
    }
    bottom = -999999;
    home();
    if (font)
        setup_font();
}

zap_pages()
{
    actually_zap_pages();
    record_zap_pages();
}

record_zap_pages()
{

```

```

    if (recording) {
        printf(".ZP\n");
    }
}

actually_zap_pages()
{
    do_zap_pages();
}

start_line()
{
    if (!in_line) {
        line_x = cur_x;
        line_y = cur_y;
        line_height = 0;
        source_pos = 0; /* Later */
        actually_start_line();
        record_start_line();
    }
}

record_start_line()
{
    if (recording) {
        printf(".SL\n");
    }
}

actually_start_line()
{
    in_line = 1;
    do_start_line();
}

end_line()
{
    flush_str();
    if (in_line) {
        source_len = 0; /* Later */
        source_name = "";
        actually_end_line(line_x, line_y, cur_x - line_x, -line_height,
                           source_name, source_pos, source_len);
        record_end_line(line_x, line_y, cur_x - line_x, -line_height,
                          source_name, source_pos, source_len);
    }
}

record_end_line(x, y, w, h, name, pos, len)
int x, y, w, h;
char *name;
int pos, len;
{
    if (recording) {
        /*
            printf("%d %d\nc~ %s~\n%d %d %d %d\n.EL\n",
                    len, pos, name, h, w, y, x);
        */
    }
}

```

```

        printf(".EL\n");
    } else {
        printf("l");
    }
}

actually_end_line(x, y, w, h, name, pos, len)
int x, y, w, h;
char *name;
int pos, len;
{
    in_line = 0;
    do_end_line(x, y, w, h, name, pos, len);
}

set_title(title)
    char *title;
{
    actually_set_pile_name(title);
    record_set_pile_name(title);
}

record_set_pile_name(name)
    char *name;
{
    if (recording) {
        printf("c~ %s~\n.SN\n", name);
    }
}

actually_set_pile_name(title)
    char *title;
{
    do_set_pile_name(title);
}

set_font(name, size)
    char *name;
    int size;
{
    if ((font = find_metrics(name, size)) == NULL) {
        fprintf(stderr, "Can't find font %s %d!\n", name, size);
        bomb(1);
    }

    setup_font();
}

setup_font()
{
    flush_str();
    if (on_page)
        use_font(font->name, font->size);
    space_width = string_width(font, " ");
    default_line_height = font->height;
}

use_font(name, size)

```

```

        char *name;
        int size;
    {
        actually_use_font(name, size);
        record_use_font(name, size);
    }

record_use_font(name, size)
    char *name;
    int size;
{
    if (recording) {
        printf("%d\\nc~ %s~\\n.UF\\n", size, name);
    }
}

actually_use_font(name, size)
    char *name;
    int size;
{
    do_use_font(name, size);
}

new_line()
{
    end_line();
    if (line_height == 0)
        line_height = default_line_height;
    cur_y -= line_height + line_space;
    cur_x = left;
    if ((cur_y - default_line_height) <= bottom) {
        end_page();
        start_page();
    }
    start_line();
}

space()
{
    if ((str_y != cur_y - font->height) || (cur_x != str_x + str_width)) {
        flush_str();
    }

    if (str_len != 0) {
        str[str_len++] = ' ';
        str[str_len] = '\\0';
        str_width += space_width;
    }
    cur_x += space_width;
}

place_image(x, y, width, height)
    int x, y, width, height;
{
    image_x = x;
    image_y = y;
    image_width = width;
    image_height = height;
}

```

```

    if (line_height < height)
        line_height = height;
}

flush_str()
{
    if (str_len) {
        str[str_len] = '\0';
        while (str_len > 0 && str[str_len-1] == ' ')
            str[--str_len] = '\0';
        actually_put_string(str, str_x, str_y + font->descent);
        record_put_string(str, str_x, str_y + font->descent);
        str_len = 0;
    }
}

record_put_string(str, x, y)
    char *str;
    int x, y;
{
    if (recording) {
        printf("%d %d\nc~ %s~\n.PS\n", y, x, str);
    }
}

actually_put_string(str, x, y)
    char *str;
    int x, y;
{
    do_put_string(str, x, y);
}

show_string(string)
    char *string;
{
    fit_image(string_width(font, string), font->height);

    stuff_string(string);
}

center_string(string)
    char *string;
{
    int width;

    width = string_width(font, string);
    cur_x = (left + right - width) >> 1;

    fit_image(width, font->height);

    stuff_string(string);
}

right_string(string)
    char *string;
{
    int width;

```

```

width = string_width(font, string);
cur_x = right - width;

fit_image(width, font->height);

stuff_string(string);
}

stuff_string(string)
    char *string;
{
    if ((image_y != str_y) || (image_x != str_x + str_width)) {
        flush_str();
    }

    if (str_len == 0) {
        str_width = 0;
        str_x = image_x;
        str_y = image_y;
    }

    str[str_len] = '\0';
    strcat(str, string);
    str_len += strlen(string);
    str_width += image_width;
}

paint_picture(name)
    char *name;
{
    ENTRY *ent = FindInIndex(Master->pictures, name);

    flush_str();

    if (ent == NULL) {
        fprintf(stderr, "Can't find image name \"%s\" in master index!\n", name);
        bomb(1);
    }

    define_picture(ent);
    fit_image(InstanceWidth(InstanceOfEntry(ent)),
              InstanceHeight(InstanceOfEntry(ent)));
    put_picture(ent, image_x, image_y);
}

put_picture(ent, x, y)
    ENTRY *ent;
    int x, y;
{
    actually_put_picture(KeyOfEntry(ent), x, y);
    record_put_picture(KeyOfEntry(ent), x, y);
}

actually_put_picture(name, x, y)
    char *name;
    int x, y;
{
    ENTRY *ent = FindInIndex(Master->pictures, name);

```

```

    if (ent == NULL) {
        fprintf(stderr, "Can't find image name \"%s\" in master index!\n", name);
        bomb(1);
    }

    define_picture(ent);
    do_put_picture(InstanceId(InstanceOfEntry(ent)), x, y);
}

record_put_picture(name, x, y)
    char *name;
    int x, y;
{
    if (recording) {
        printf("%d %d\nc~ %s~\n.PP\n", y, x, name);
    }
}

fit_image(width, height)
    int width, height;
{
    if ((cur_x + width > right) && (cur_x != left))
        new_line();

    if (((cur_y - height) <= bottom) && (cur_y != top)) {
        end_page();
        start_page();
    }

    place_image(cur_x, cur_y - height, width, height);

    cur_x += width;
}

stamp_target(name, ref)
    char *name;
    char *ref;
{
    ENTRY *ent = FindInIndex(Master->targets, name);

    flush_str();

    if (ent == NULL) {
        fprintf(stderr, "Can't find target name \"%s\" in master index!\n", name);
        bomb(1);
    }

    define_target(ent);

    place_image(image_x, image_y,
        InstanceWidth(InstanceOfEntry(ent)) ?
            InstanceWidth(InstanceOfEntry(ent)) : image_width,
        InstanceHeight(InstanceOfEntry(ent)) ?
            InstanceHeight(InstanceOfEntry(ent)) : image_height);

    put_target(ent, ref, image_x, image_y, image_width, image_height);
}

```

```

put_target(ent, ref, x, y, w, h)
    ENTRY *ent;
    char *ref;
    int x, y, w, h;
{
    actually_put_target(KeyOfEntry(ent), ref, x, y, w, h);
    record_put_target(KeyOfEntry(ent), ref, x, y, w, h);
}

record_put_target(name, ref, x, y, w, h)
    char *name, *ref;
    int x, y, w, h;
{
    if (recording) {
        printf("%d %d %d %d\nc~ %s~\nc~ %s~\n.PT\n",
            h, w, y, x, ref, name);
    }
}

actually_put_target(name, ref, x, y, w, h)
    char *name, *ref;
    int x, y, w, h;
{
    ENTRY *ent = FindInIndex(Master->targets, name);

    if (ent == NULL) {
        fprintf(stderr, "Can't find target name \"%s\" in master index!\n", name);
        bomb(1);
    }

    define_target(ent);

    do_put_target(InstanceId(InstanceOfEntry(ent)), ref,
        x, y, w, h);
}

stamp_background_target(name, ref)
    char *name;
    char *ref;
{
    ENTRY *ent = FindInIndex(Master->targets, name);

    flush_str();

    if (ent == NULL) {
        fprintf(stderr, "Can't find target name \"%s\" in master index!\n", name);
        bomb(1);
    }

    define_target(ent);

    place_image(0, 0, page_width, page_height);

    put_target(ent, ref, image_x, image_y, image_width, image_height);
}

free_stamps()

```



```

{
    FreeInstances();
    do_free_stamps();
}

setup_definition_pile()
{
    do_setup_definition_pile();
}

setup_controls_pile()
{
    do_setup_controls_pile();
}

setup_contents_pile()
{
    do_setup_contents_pile();
}

use_pile(pileid)
    int pileid;
{
    do_use_pile(pileid);
}

name_pile(name)
    char *name;
{
    do_name_pile(name);
}

int use_linked_pile(name)
    char *name;
{
    int id;

    record_use_linked_pile(name);
    return(actually_use_linked_pile(name));
}

int actually_use_linked_pile(name)
    char *name;
{
    int id;

    do_use_linked_pile(name, &id);
    return(id);
}

record_use_linked_pile(name)
    char *name;
{
    if (recording) {
        printf("c~ %s~\n.ULP\n", name);
    }
}

```

```

int use_parent_pile()
{
    int id;

    record_use_parent_pile();
    return(actually_use_parent_pile());
}

int actually_use_parent_pile()
{
    int id;

    do_use_parent_pile(&id);
    return(id);
}

record_use_parent_pile()
{
    if (recording) {
        printf(".UPP\n");
    }
}

def_local(name)
    char *name;
{
    do_def_local(name);
}

def_global(name)
    char *name;
{
    do_def_global(name);
}

write_ps_char(c)
    char c;
{
    psio_putc(c, PostScript);
}

write_ps_text(addr, len)
    char *addr;
    int len;
{
    while (--len >= 0) {
        psio_putc(*addr++, PostScript);
    }
}

send_ps_text(addr)
    char *addr;
{
    write_ps_text(addr, strlen(addr));
}

send_ps_string(str)
    char *str;

```

```

{
    do_send_ps_string(str);
}

send_ps_int(i)
    int i;
{
    do_send_ps_int(i);
}

/*
 * Initialization stuff
 */

FILE *MasterIndexFile;

init_fmt()
{
    char pathname[MAXPATHLEN]; /* From <sys/param.h> */

    cached_metrics = NULL;
    PostScriptInput = PostScript = NULL;

    if ((MasterIndexFile = fopen("master-index", "r")) == NULL) {
        fprintf(stderr, "Cannot open master-index file!\n");
        bomb(1);
    }

    Master = ReadMasterIndex(CreateMasterIndex(), MasterIndexFile);

    fclose(MasterIndexFile);

    if (ps_open_PostScript() == 0) {
        fprintf(stderr, "Cannot connect to NeWS server!\n");
        bomb(1);
    }

    do_initialize(getwd(pathname));
}

flush_PS()
{
    ps_flush_PostScript();
}

quit_PS()
{
    ps_close_PostScript();
}

no_forth()
{
    fprintf(stderr, "Bombing! (no forth!)\n");
    exit(1);
}

int (*forth_bomb)() = no_forth;
int (*forth_exec)() = no_forth;

```

```
bomb()  
{  
    fflush(stdout);  
    fflush(stderr);  
    forth_bomb();  
}
```