```
/* Printing/Printing
 *
 * GtkPrintOperation offers a simple API to support printing
 * in a cross-platform way.
 *
 */

#include <math.h>
#include <gtk/gtk.h>

/* In points */
#define HEADER_HEIGHT (10*72/25.4)
#define HEADER_GAP (3*72/25.4)

typedef struct
{
  gchar *resourcename;
  gdouble font_size;

  gint lines_per_page;
  gchar **lines;
  gint num_lines;
  gint num_pages;
} PrintData;

static void
begin_print (GtkPrintOperation *operation,
             GtkPrintContext   *context,
             gpointer           user_data)
{
  PrintData *data = (PrintData *)user_data;
  GBytes *bytes;
  int i;
  double height;

  height = gtk_print_context_get_height (context) - HEADER_HEIGHT - HEADER_GAP;

  data->lines_per_page = floor (height / data->font_size);

  bytes = g_resources_lookup_data (data->resourcename, 0, NULL);

  data->lines = g_strsplit (g_bytes_get_data (bytes, NULL), "\n", 0);
  g_bytes_unref (bytes);

  i = 0;
  while (data->lines[i] != NULL)
    i++;

  data->num_lines = i;
  data->num_pages = (data->num_lines - 1) / data->lines_per_page + 1;

  gtk_print_operation_set_n_pages (operation, data->num_pages);
}

static void
draw_page (GtkPrintOperation *operation,
           GtkPrintContext   *context,
           gint               page_nr,
           gpointer           user_data)
{
  PrintData *data = (PrintData *)user_data;
  cairo_t *cr;
```

```
PangoLayout *layout;
gint text_width, text_height;
gdouble width;
gint line, i;
PangoFontDescription *desc;
gchar *page_str;

cr = gtk_print_context_get_cairo_context (context);
width = gtk_print_context_get_width (context);

cairo_rectangle (cr, 0, 0, width, HEADER_HEIGHT);

cairo_set_source_rgb (cr, 0.8, 0.8, 0.8);
cairo_fill_preserve (cr);

cairo_set_source_rgb (cr, 0, 0, 0);
cairo_set_line_width (cr, 1);
cairo_stroke (cr);

layout = gtk_print_context_create_pango_layout (context);

desc = pango_font_description_from_string ("sans 14");
pango_layout_set_font_description (layout, desc);
pango_font_description_free (desc);

pango_layout_set_text (layout, data->resourcename, -1);
pango_layout_get_pixel_size (layout, &text_width, &text_height);

if (text_width > width)
  {
    pango_layout_set_width (layout, width);
    pango_layout_set_ellipsize (layout, PANGO_ELLIPSIZE_START);
    pango_layout_get_pixel_size (layout, &text_width, &text_height);
  }

cairo_move_to (cr, (width - text_width) / 2,  (HEADER_HEIGHT - text_height) / 2);
pango_cairo_show_layout (cr, layout);

page_str = g_strdup_printf ("%d/%d", page_nr + 1, data->num_pages);
pango_layout_set_text (layout, page_str, -1);
g_free (page_str);

pango_layout_set_width (layout, -1);
pango_layout_get_pixel_size (layout, &text_width, &text_height);
cairo_move_to (cr, width - text_width - 4, (HEADER_HEIGHT - text_height) / 2);
pango_cairo_show_layout (cr, layout);

g_object_unref (layout);

layout = gtk_print_context_create_pango_layout (context);

desc = pango_font_description_from_string ("monospace");
pango_font_description_set_size (desc, data->font_size * PANGO_SCALE);
pango_layout_set_font_description (layout, desc);
pango_font_description_free (desc);

cairo_move_to (cr, 0, HEADER_HEIGHT + HEADER_GAP);
line = page_nr * data->lines_per_page;
for (i = 0; i < data->lines_per_page && line < data->num_lines; i++)
  {
    pango_layout_set_text (layout, data->lines[line], -1);
    pango_cairo_show_layout (cr, layout);
```

```
      cairo_rel_move_to (cr, 0, data->font_size);
      line++;
    }

  g_object_unref (layout);
}

static void
end_print (GtkPrintOperation *operation,
           GtkPrintContext   *context,
           gpointer           user_data)
{
  PrintData *data = (PrintData *)user_data;

  g_free (data->resourcename);
  g_strfreev (data->lines);
  g_free (data);
}


GtkWidget *
do_printing (GtkWidget *do_widget)
{
  GtkPrintOperation *operation;
  GtkPrintSettings *settings;
  PrintData *data;
  GError *error = NULL;

  operation = gtk_print_operation_new ();
  data = g_new0 (PrintData, 1);
  data->resourcename = g_strdup ("/sources/printing.c");
  data->font_size = 12.0;

  g_signal_connect (G_OBJECT (operation), "begin-print",
                    G_CALLBACK (begin_print), data);
  g_signal_connect (G_OBJECT (operation), "draw-page",
                    G_CALLBACK (draw_page), data);
  g_signal_connect (G_OBJECT (operation), "end-print",
                    G_CALLBACK (end_print), data);

  gtk_print_operation_set_use_full_page (operation, FALSE);
  gtk_print_operation_set_unit (operation, GTK_UNIT_POINTS);
  gtk_print_operation_set_embed_page_setup (operation, TRUE);

  settings = gtk_print_settings_new ();

  gtk_print_settings_set (settings, GTK_PRINT_SETTINGS_OUTPUT_BASENAME, "gtk-demo")
  gtk_print_operation_set_print_settings (operation, settings);

  gtk_print_operation_run (operation, GTK_PRINT_OPERATION_ACTION_PRINT_DIALOG, GTK_

  g_object_unref (operation);
  g_object_unref (settings);

  if (error)
    {
      GtkWidget *dialog;

      dialog = gtk_message_dialog_new (GTK_WINDOW (do_widget),
                                       GTK_DIALOG_DESTROY_WITH_PARENT,
                                       GTK_MESSAGE_ERROR,
                                       GTK_BUTTONS_CLOSE,
```

```
                                          "%s", error->message);
      g_error_free (error);

      g_signal_connect (dialog, "response",
                        G_CALLBACK (gtk_widget_destroy), NULL);

      gtk_widget_show (dialog);
    }


  return NULL;
}
```