GUI programming using GTK

dumping things from my1gtktest_single...

A useful makefile. To compile more.c that uses GTK2 (default is GTK3), run `make APP_NAME=more GTK_VERS=2.0`.

makefile

```
# makefile for simple gtk application written in single c source file

ALLAPP = $(subst .c,,$(subst src/,,$(wildcard src/*.c)))
ALLAPP += $(subst .c,,$(wildcard *.c))

GTKVER ?= 3.0

CFLAGS += $(shell pkg-config --cflags gtk+-$(GTKVER))
LFLAGS += $(shell pkg-config --libs gtk+-$(GTKVER))

.PHONY: dummy $(TARGET)

dummy:
    @echo
    @echo "Run 'make <app>' or 'make <app> GTKVER=2.0'"
    @echo "  <app> = { $(ALLAPP) }"
    @echo

%: %.c
    gcc -Wall $(CFLAGS) -o $@ $+ $(LFLAGS)

clean:
    rm -rf $(ALLAPP)
```

Sample Codes

tray.c

```
/*-------------------------------------------------------------------------
-------*/
#include <gtk/gtk.h>
/*-------------------------------------------------------------------------
-------*/
void tray_icon_on_click(GtkStatusIcon *status_icon, gpointer user_data)
{
    printf("Clicked on tray icon\n");
}
```

```
/*-------------------------------------------------------------------------
-------*/
void tray_icon_on_menu(GtkStatusIcon *status_icon, guint button,
    guint activate_time, gpointer user_data)
{
    printf("Popup menu\n");
    gtk_main_quit();
}
/*-------------------------------------------------------------------------
-------*/
static GtkStatusIcon *create_tray_icon()
{
    GtkStatusIcon *tray_icon;
    tray_icon = gtk_status_icon_new();
    g_signal_connect(G_OBJECT(tray_icon), "activate",
        G_CALLBACK(tray_icon_on_click), NULL);
    g_signal_connect(G_OBJECT(tray_icon), "popup-menu",
        G_CALLBACK(tray_icon_on_menu), NULL);
    gtk_status_icon_set_from_icon_name(tray_icon,
GTK_STOCK_MEDIA_STOP);
    gtk_status_icon_set_tooltip(tray_icon, "Example Tray Icon");
    gtk_status_icon_set_visible(tray_icon, TRUE);
    return tray_icon;
}
/*-------------------------------------------------------------------------
-------*/
int main(int argc, char **argv)
{
    GtkStatusIcon *tray_icon;
    gtk_init(&argc, &argv);
    tray_icon = create_tray_icon();
    printf("embedded: ");
    if (!gtk_status_icon_is_embedded(tray_icon))
    {
        printf("yes\n");
        gtk_main();
        printf("done\n");
    }
    else
    {
        printf("no\nCannot embed tray icon - aborting!\n");
    }
    return 0;
}
/*-------------------------------------------------------------------------
-------*/
```

draw.c

```
/*-------------------------------------------------------------------------
```

```
-------*/
#include <gtk/gtk.h>
/**
 * From,
 * https://developer.gnome.org/gtk3/stable/ch01s05.html
 *
 * - my rewrite to understand these new gtk3 stuffs (cairo surface! :o)
**/
/*--------------------------------------------------------------------------
-------*/
#define MY1APP_FULL "org.my1.gtktest"
#define MY1APP_NAME "GTK Test"
/*--------------------------------------------------------------------------
-------*/
typedef struct _my1gtk_app_t
{
    int stat;
    cairo_surface_t *surf;
    GtkApplication *main;
    GtkWidget *gwin;
    GtkWidget *view;
    GtkWidget *draw;
}
my1gtk_app_t;
/*--------------------------------------------------------------------------
-------*/
void my1gtk_app_destroy(gpointer data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    if (gapp->surf)
        cairo_surface_destroy(gapp->surf);
}
/*--------------------------------------------------------------------------
-------*/
gboolean my1gtk_app_draw_cb(GtkWidget *widget,cairo_t *cr,gpointer
data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    cairo_set_source_surface(cr,gapp->surf,0,0);
    cairo_paint(cr);
    return FALSE;
}
/*--------------------------------------------------------------------------
-------*/
void clear_surface(my1gtk_app_t* gapp)
{
    cairo_t *cr  = cairo_create(gapp->surf);
    cairo_set_source_rgb (cr, 1, 1, 1);
    cairo_paint(cr);
    cairo_destroy(cr);
}
```

```
/*------------------------------------------------------------------------
-------*/
gboolean my1gtk_app_conf_cb(GtkWidget *widget,
    GdkEventConfigure *event, gpointer data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    if (gapp->surf)
        cairo_surface_destroy(gapp->surf);
    gapp->surf = gdk_window_create_similar_surface(
        gtk_widget_get_window(widget), CAIRO_CONTENT_COLOR,
        gtk_widget_get_allocated_width(widget),
        gtk_widget_get_allocated_height (widget));
        clear_surface(gapp);
    return TRUE;
}
/*------------------------------------------------------------------------
-------*/
void draw_brush(GtkWidget *widget, gdouble x, gdouble y, gpointer data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    cairo_t *cr = cairo_create (gapp->surf);
    cairo_rectangle (cr,x-3,y-3,6,6);
    cairo_fill (cr);
    cairo_destroy (cr);
    gtk_widget_queue_draw_area(widget,x-3,y-3,6,6);
}
/*------------------------------------------------------------------------
-------*/
gboolean my1gtk_app_motion_cb(GtkWidget *widget,
    GdkEventMotion *event, gpointer data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    if (!gapp->surf)
        return FALSE;
    if (event->state&GDK_BUTTON1_MASK)
        draw_brush(widget,event->x,event->y,data);
    return TRUE;
}
/*------------------------------------------------------------------------
-------*/
gboolean my1gtk_app_bpress_cb(GtkWidget *widget,
    GdkEventButton *event, gpointer data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    if (!gapp->surf)
        return FALSE;
    if (event->button == GDK_BUTTON_PRIMARY)
        draw_brush (widget,event->x,event->y,data);
    else if (event->button == GDK_BUTTON_SECONDARY)
    {
        clear_surface(gapp);
```

```
        gtk_widget_queue_draw(widget);
    }
    return TRUE;
}
/*------------------------------------------------------------------------
-------*/
void my1gtk_app_activate(GtkApplication *app, gpointer data)
{
    my1gtk_app_t* gapp = (my1gtk_app_t*)data;
    /** app === gapp->main! assert?*/
    gapp->gwin = gtk_application_window_new(app);
    gtk_window_set_title(GTK_WINDOW(gapp->gwin),MY1APP_NAME);
    g_signal_connect_swapped(gapp->gwin,"destroy",
        G_CALLBACK(my1gtk_app_destroy),data);
    gtk_container_set_border_width(GTK_CONTAINER(gapp->gwin),4);
    gapp->view = gtk_frame_new(0x0);
    gtk_frame_set_shadow_type(GTK_FRAME(gapp->view),GTK_SHADOW_IN);
    gtk_container_add(GTK_CONTAINER(gapp->gwin),gapp->view);
    gapp->draw = gtk_drawing_area_new();
    /* set a minimum size */
    gtk_widget_set_size_request(gapp->draw,100,100);
    gtk_container_add(GTK_CONTAINER(gapp->view),gapp->draw);
    g_signal_connect(gapp->draw,"draw",
        G_CALLBACK(my1gtk_app_draw_cb),data);
    g_signal_connect(gapp->draw,"configure-event",
        G_CALLBACK(my1gtk_app_conf_cb),data);
    g_signal_connect(gapp->draw,"motion-notify-event",
        G_CALLBACK(my1gtk_app_motion_cb),data);
    g_signal_connect(gapp->draw,"button-press-event",
        G_CALLBACK(my1gtk_app_bpress_cb),data);
    gtk_widget_set_events(gapp->draw,gtk_widget_get_events(gapp->draw)
        | GDK_BUTTON_PRESS_MASK | GDK_POINTER_MOTION_MASK);
    gtk_widget_show_all(gapp->gwin);
}
/*------------------------------------------------------------------------
-------*/
void my1gtk_app_init(my1gtk_app_t* gapp)
{
    gapp->stat = 0;
    gapp->surf = 0x0;
    gapp->main =
gtk_application_new(MY1APP_FULL,G_APPLICATION_FLAGS_NONE);
    g_signal_connect(gapp->main,"activate",
        G_CALLBACK(my1gtk_app_activate),(gpointer)gapp);
}
/*------------------------------------------------------------------------
-------*/
void my1gtk_app_exec(my1gtk_app_t* gapp, int argc, char* argv[])
{
    gapp->stat =
g_application_run(G_APPLICATION(gapp->main),argc,argv);
```

```
    g_object_unref(gapp->main);
}
/*------------------------------------------------------------------------
-------*/
int main (int argc, char **argv)
{
    my1gtk_app_t gapp;
    my1gtk_app_init(&gapp);
    my1gtk_app_exec(&gapp,argc,argv);
    return gapp.stat;
}
/*------------------------------------------------------------------------
-------*/
```

testcss.c

```
/*------------------------------------------------------------------------
-------*/
#include <gtk/gtk.h>
/*------------------------------------------------------------------------
-------*/
#define MY1APP_FULL "org.my1.gtktest"
#define MY1APP_NAME "GTK Test"
/*------------------------------------------------------------------------
-------*/
GtkCssProvider *ccss;
/*------------------------------------------------------------------------
-------*/
gboolean on_show_resize(GtkWidget *widget, GtkAllocation *allocation,
    gpointer user_data)
{
    char test[] = "GtkLabel { background-color: #AAAAAA; font-size:
%.0fpx; }";
    char buff[128];
    snprintf(buff,128,test,((float)allocation->height*12.0/40.0));
    printf("Resize: %d x %d =>
%s\n",allocation->width,allocation->height,buff);
    gtk_css_provider_load_from_data(ccss,buff, -1, NULL);
/**
    my1image_view_t* view = (my1image_view_t*) user_data;
    printf("Resize: %d x %d\n",allocation->width,allocation->height);
*/
    return TRUE;
}
/*------------------------------------------------------------------------
-------*/
static void activate(GtkApplication* app, gpointer data)
{
    GtkWidget *gwin, *grid, *show;
    GtkCssProvider *gcss;
```

```
    GdkDisplay *disp;
    GdkScreen *gscr;
    gwin = gtk_application_window_new(app);
    gtk_window_set_title(GTK_WINDOW(gwin),MY1APP_FULL);
    grid = gtk_grid_new();
    gtk_container_add(GTK_CONTAINER(gwin),grid);
    show = gtk_label_new("HAHA");
    gtk_widget_set_hexpand(show,TRUE);
    gtk_widget_set_halign(show,GTK_ALIGN_FILL);
    gtk_widget_set_vexpand(show,TRUE);
    gtk_widget_set_valign(show,GTK_ALIGN_FILL);
    gtk_grid_attach(GTK_GRID(grid),show,0,0,1,1);
    gcss = gtk_css_provider_new();
    disp = gdk_display_get_default();
    gscr = gdk_display_get_default_screen(disp);
    gtk_style_context_add_provider_for_screen(gscr,
        GTK_STYLE_PROVIDER(gcss),GTK_STYLE_PROVIDER_PRIORITY_USER);
    gtk_css_provider_load_from_data(gcss,
        "GtkLabel { background-color: #AAAAAA;}\n"
        "GtkGrid  { background-color: #4444AA;}", -1, NULL);
    ccss = gtk_css_provider_new();
    gtk_style_context_add_provider_for_screen(gscr,
        GTK_STYLE_PROVIDER(ccss),GTK_STYLE_PROVIDER_PRIORITY_USER);
    g_signal_connect(G_OBJECT(show),"size-allocate",
        G_CALLBACK(on_show_resize),0x0);
    /**gtk_window_fullscreen(GTK_WINDOW(gwin));*/
    gtk_widget_show_all(gwin);
}
/*------------------------------------------------------------------------
-------*/
int main(int argc, char **argv)
{
    int stat;
    GtkApplication *gapp;
    gapp = gtk_application_new(MY1APP_FULL,G_APPLICATION_FLAGS_NONE);
    g_signal_connect(gapp,"activate",G_CALLBACK(activate),NULL);
    stat = g_application_run(G_APPLICATION(gapp),argc,argv);
    g_object_unref(gapp);
    return stat;
}
/*------------------------------------------------------------------------
-------*/
```