#include "my\_application.h"

#include <flutter\_linux/flutter\_linux.h>

#ifdef GDK\_WINDOWING\_X11

#include <gdk/gdkx.h>

#endif

#include "flutter/generated\_plugin\_registrant.h"

struct \_MyApplication {

GtkApplication parent\_instance;

char\*\* dart\_entrypoint\_arguments;

};

G\_DEFINE\_TYPE(MyApplication, my\_application, GTK\_TYPE\_APPLICATION)

// Implements GApplication::activate.

static void my\_application\_activate(GApplication\* application) {

MyApplication\* self = MY\_APPLICATION(application);

GtkWindow\* window =

GTK\_WINDOW(gtk\_application\_window\_new(GTK\_APPLICATION(application)));

// Use a header bar when running in GNOME as this is the common style used

// by applications and is the setup most users will be using (e.g. Ubuntu

// desktop).

// If running on X and not using GNOME then just use a traditional title bar

// in case the window manager does more exotic layout, e.g. tiling.

// If running on Wayland assume the header bar will work (may need changing

// if future cases occur).

gboolean use\_header\_bar = TRUE;

#ifdef GDK\_WINDOWING\_X11

GdkScreen\* screen = gtk\_window\_get\_screen(window);

if (GDK\_IS\_X11\_SCREEN(screen)) {

const gchar\* wm\_name = gdk\_x11\_screen\_get\_window\_manager\_name(screen);

if (g\_strcmp0(wm\_name, "GNOME Shell") != 0) {

use\_header\_bar = FALSE;

}

}

#endif

if (use\_header\_bar) {

GtkHeaderBar\* header\_bar = GTK\_HEADER\_BAR(gtk\_header\_bar\_new());

gtk\_widget\_show(GTK\_WIDGET(header\_bar));

gtk\_header\_bar\_set\_title(header\_bar, "moneyexchanger");

gtk\_header\_bar\_set\_show\_close\_button(header\_bar, TRUE);

gtk\_window\_set\_titlebar(window, GTK\_WIDGET(header\_bar));

} else {

gtk\_window\_set\_title(window, "moneyexchanger");

}

gtk\_window\_set\_default\_size(window, 1280, 720);

gtk\_widget\_show(GTK\_WIDGET(window));

g\_autoptr(FlDartProject) project = fl\_dart\_project\_new();

fl\_dart\_project\_set\_dart\_entrypoint\_arguments(project, self->dart\_entrypoint\_arguments);

FlView\* view = fl\_view\_new(project);

gtk\_widget\_show(GTK\_WIDGET(view));

gtk\_container\_add(GTK\_CONTAINER(window), GTK\_WIDGET(view));

fl\_register\_plugins(FL\_PLUGIN\_REGISTRY(view));

gtk\_widget\_grab\_focus(GTK\_WIDGET(view));

}

// Implements GApplication::local\_command\_line.

static gboolean my\_application\_local\_command\_line(GApplication\* application, gchar\*\*\* arguments, int\* exit\_status) {

MyApplication\* self = MY\_APPLICATION(application);

// Strip out the first argument as it is the binary name.

self->dart\_entrypoint\_arguments = g\_strdupv(\*arguments + 1);

g\_autoptr(GError) error = nullptr;

if (!g\_application\_register(application, nullptr, &error)) {

g\_warning("Failed to register: %s", error->message);

\*exit\_status = 1;

return TRUE;

}

g\_application\_activate(application);

\*exit\_status = 0;

return TRUE;

}

// Implements GObject::dispose.

static void my\_application\_dispose(GObject\* object) {

MyApplication\* self = MY\_APPLICATION(object);

g\_clear\_pointer(&self->dart\_entrypoint\_arguments, g\_strfreev);

G\_OBJECT\_CLASS(my\_application\_parent\_class)->dispose(object);

}

static void my\_application\_class\_init(MyApplicationClass\* klass) {

G\_APPLICATION\_CLASS(klass)->activate = my\_application\_activate;

G\_APPLICATION\_CLASS(klass)->local\_command\_line = my\_application\_local\_command\_line;

G\_OBJECT\_CLASS(klass)->dispose = my\_application\_dispose;

}

static void my\_application\_init(MyApplication\* self) {}

MyApplication\* my\_application\_new() {

return MY\_APPLICATION(g\_object\_new(my\_application\_get\_type(),

"application-id", APPLICATION\_ID,

"flags", G\_APPLICATION\_NON\_UNIQUE,

nullptr));

}