Writing a GNOME application for newcomers

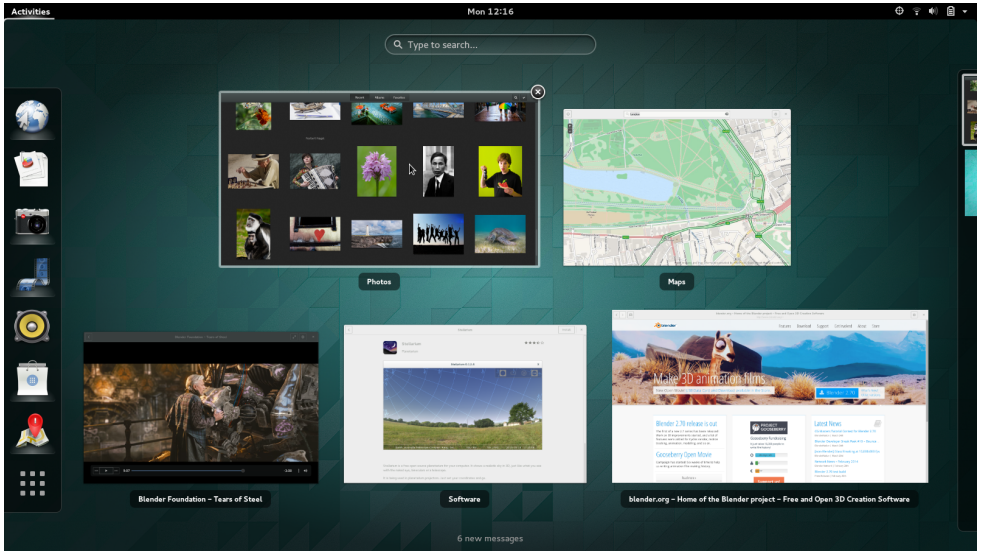
David King <amigadave@amigadave.com>

21st April 2016 / GNOME.Asia / workshop

Licensed under CC0-1.0

http://amigadave.fedorapeople.org/gnome_asia_training_2016.pdf

Overview



- GNOME as a platform for application development
- 6-month development cycle
- write applications in Python, JavaScript, Vala or C (or others)

Platform libraries

- GLib (including GObject and GIO)
- GTK+
- many others (see the application development overview on developer.gnome.org)

Development tools

- Glade, for designing GUIs
- Devhelp, for browsing API reference documentation
- Builder, a future GNOME IDE
- Inspector, a GTK+ tool for debugging UIs

Before we start

- Clone the git repository: `git clone git://fedorapeople.org/home/fedora/amigadave/public_git/c-gnome-app.git`
- Browse through the git commits: `http://fedorapeople.org/cgit/amigadave/public_git/c-gnome-app.git/`
- Open your favourite text editor or IDE
- Build: `autoreconf -f -i; ./configure; make`
- Try running the application: `./c-gnome-app`

Hello world!

- Include gtk/gtk.h
- Initialize GTK+
- Show a window
- Run the GTK+ main loop
- The application must be killed externally!

Hello world code

```c
#include <gtk/gtk.h>

int main (int argc, char *argv[])
{
    GtkWidget *window;
    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_show_all (window);
    gtk_main ();
    return 0;
}
```

Signals and handlers

- Connect the `activate` signal of the application to a handler
- Create or show the window in the handler
- GTK+ widgets (and other `GObject`s) have signals, which are documented in the API references
- The application terminates when the window is closed
- See https://wiki.gnome.org/HowDoI/GtkApplication for more details on GtkApplication

Signals and handlers code 1

```c
#include <gtk/gtk.h>

static void
on_activate (GApplication *app,
             gpointer user_data)
{
    GtkWidget *window;

    window = gtk_application_window_new (
                                        GTK_APPLICATION (app));
    gtk_widget_show_all (window);
}

/* Continued on next slide. */
```

Signals and handlers code 2

```
/* Continued from previous slide. */

int main (int argc, char *argv[])
{
    GtkApplication *app;
    gint status;
    app = gtk_application_new ("org.example.CGnome",
                               G_APPLICATION_FLAGS_NONE);
    g_signal_connect (app, "activate",
                      G_CALLBACK (on_activate), NULL);
    status = g_application_run (G_APPLICATION (app),
                                argc, argv);
    g_object_unref (app);
    return status;
}
```

Keybindings and actions

- Add an action for quitting the application, and another for printing hello world
- Connect the `activate` signal of the actions to handlers
- Associate an accelerator with each action
- See https://wiki.gnome.org/HowDoI/GAction for more details

Keybindings and actions code 1

```c
#include <gtk/gtk.h>

static GtkWidget *window;

static void
on_hello_world (GSimpleAction *action,
                GVariant *parameter,
                gpointer user_data)
{
    g_print ("%s\n", "Hello world!");
}

static void
on_quit (GSimpleAction *action,
         GVariant *parameter,
         gpointer user_data)
{
    g_application_quit (G_APPLICATION (user_data));
}

/* Continued on next slide. */
```

Keybindings and actions code 2

```
/* Continued from previous slide. */
static const GActionEntry actions[] =
{
    { "hello-world", on_hello_world },
    { "quit", on_quit }
};

static void
on_activate (GApplication *app,
             gpointer user_data)
{
    gtk_widget_show_all (window);
}
/* Continued on next slide. */
```

Keybindings and actions code 3

```
/* Continued from previous slide. */
static void on_startup (GApplication *app,
                        gpointer user_data)
{
 const gchar * const hello_world_accel[] = { "<Primary>h",
                                             NULL };
 const gchar * const quit_accel[] = { "<Primary>q", NULL };
 g_action_map_add_action_entries (G_ACTION_MAP (app),
                                  actions,
                                  G_N_ELEMENTS (actions),
                                  app);
 window = gtk_application_window_new (GTK_APPLICATION (app));
 gtk_application_set_accels_for_action (GTK_APPLICATION (app),
                                        "app.hello-world",
                                        hello_world_accel);
 gtk_application_set_accels_for_action (GTK_APPLICATION (app),
                                        "app.quit",
                                        quit_accel);
}
/* Continued on next slide. */
```

Keybindings and actions code 4

```
/* Continued from previous slide. */
int main (int argc, char *argv[])
{
    GtkApplication *app;
    gint status;
    app = gtk_application_new ("org.example.CGnome",
                               G_APPLICATION_FLAGS_NONE);
    g_signal_connect (app, "activate",
                      G_CALLBACK (on_activate), NULL);
    g_signal_connect (app, "startup",
                      G_CALLBACK (on_startup), NULL);
    status = g_application_run (G_APPLICATION (app),
                                argc, argv);
    g_object_unref (app);
    return status;
}
```

Application menus

- Create a menu model
- Link the menu items to actions, in the correct group
- Set the application menu on the application
- See https://wiki.gnome.org/HowDoI/ApplicationMenu for more details

Application menus code

```
/* Incomplete snippet. */
GMenu *appmenu;
GMenu *section;
GMenuItem *item;
appmenu = g_menu_new ();
section = g_menu_new ();
item = g_menu_item_new ("Hello␣world!", "app.hello-world");
g_menu_append_section (appmenu, NULL, G_MENU_MODEL (section));
g_menu_append_item (section, item);
g_object_unref (item);
item = g_menu_item_new ("Quit", "app.quit");
g_menu_append_item (section, item);
g_object_unref (item);
gtk_application_set_app_menu (gtk_app,
                              G_MENU_MODEL (appmenu));
g_object_unref (appmenu);
```

Buttons and actionable widgets

- As buttons implement the `GtkActionable` interface, they can also be linked to actions
- Set the action name on the `GtkActionable` with the `action-name` property
- As GtkWindow is a `GtkContainer`, use the `add()` method to put the button in the window
- See the GtkActionable API reference for more details

Buttons and actionable widgets code changes

```
/* Put these code lines in the right place. */

button = gtk_button_new_with_label ("Hello␣world!");
gtk_actionable_set_action_name (GTK_ACTIONABLE (button),
                                "app.hello-world");
gtk_container_add (GTK_CONTAINER (window), button);
```

A simple text editor

- Add a `GtkEntry` (or a `GtkTreeView` if you are feeling confident)
- Save the contents of the text entry when quitting, load them on startup
- No hints this time, you have to do it yourself!
- You will find the `GtkEntry` API reference helpful. Use GLib or stdio functions to load and save the text file

Deploying your application

- Install a desktop file and icon to show your application alongside others
- Use a standard build system to make your application a releasable unit
- Make regular releases, so that your application can be easily consumed
- Package your application for distributions
- Look forward to a future of application sandboxing (see my xdg-app talk during the conference)

Example desktop file

```
[Desktop]
Name=My C App
Comment=Short description of this application
Keywords=c;editor;
Type=Application
Exec=c-gnome-app
Icon=c-gnome-app
Categories=Gtk;GNOME;Utility;
```

Example autotools build system (configure.ac)

```
AC_INIT([C GNOME App],
        [0.1],
        [amigadave@amigadave.com],
        [c-gnome-app],
        [http://fedorapeople.org/cgit/amigadave/public_git/
c-gnome-app.git/])

AM_INIT_AUTOMAKE([1.11 foreign])
PKG_CHECK_MODULES([APP], [gtk+-3.0 >= 3.12])
AC_PROG_CC
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Example autotools build system (Makefile.am)

```
bin_PROGRAMS = c-gnome-app

c_gnome_app_CFLAGS = $(APP_CFLAGS)
c_gnome_app_LDFLAGS = $(APP_LIBS)
c_gnome_app_SOURCES = c-gnome-app.c

desktopdir = $(datadir)/applications
dist_desktop_DATA = c-gnome-app.desktop
```

Using the autotools build system

- Run `autoreconf --force --install` to generate the build files
- Run `./configure` to configure the project and check for required dependencies
- Run `make` to build the project, and `make install` to install it into the default prefix
- Run `make distcheck` to create a tarball and perform a build and install check

Translations

- GNOME applications use GNU gettext for marking, extracting and retrieving translatable strings
- intltool is currently used for translating desktop files and GSettings schemas, but the latest version of gettext can do this too
- See the translation guide in the application development overview

User help

- GNOME application documentation is written in the Mallard format
- Designed to be concise and task-based
- Attend the “GNOME Documentation” talks for more information
- See the user help section of the application development overview

Further resources

- Mailing lists, https://mail.gnome.org/
- Wiki, https://wiki.gnome.org/
- IRC, https://wiki.gnome.org/Community/GettingInTouch/IRC
- https://developer.gnome.org/

Settings management

- `GSettings` is the API in GIO used for storing user preferences
- Settings are stored in dconf on Unix, the registry on Windows
- `GAction` can be backed by a setting

New widgets and features

- popovers, header bars, and lots more!
- DBusActivatable applications
- widget templates
- GResource (application resources and data built into a single binary)
- xdg-app bundle