A beginner’s guide to GNOME 3 application development

David King <amigadave@amigadave.com>

23rd May 2014 / GNOME.Asia / #8 conference room
Licensed under CC0-1.0
Overview

- GNOME as a platform for application development
- 6-month development cycle
- write applications in Python, JavaScript, Vala or C
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
- Anjuta, an integrated development environment (IDE)
- 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/python-gnome-app.git`
- Browse through the git commits: `http://fedorapeople.org/cgit/amigadave/public_git/python-gnome-app.git/`
- Open your favourite text editor or IDE
- Try running the application: `./python-gnome-app`
Hello world!

- Import GObject-Introspection
- Show a window
- Run the GTK+ main loop
- The application must be killed externally!
Hello world code

```python
#!/usr/bin/python3

from gi.repository import Gtk

window = Gtk.Window()
window.show_all()

Gtk.main()
```
Signals and handlers

- **Subclass** (inherit) `GtkApplication`
- Connect the `activate` signal of the application to a handler
- Create or show the window in the handler
- GTK+ widgets (and other `GObjects`) 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`
#!/usr/bin/python3

import sys
from gi.repository import Gtk

class PythonApp(Gtk.Application):

    def __init__(self):
        Gtk.Application.__init__(self,
            application_id="org.example.PythonGnome")

        self.connect("activate", self.on_activate)

    def on_activate(self, app):
        window = Gtk.ApplicationWindow(application=app)
        window.show_all()

# Continued on next slide.
Signals and handlers code 2

# Continued from previous slide.

```python
app = PythonApp()
exit_status = app.run(None)
sys.exit(exit_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
- Add an accelerator to each action
- See https://wiki.gnome.org/HowDoI/GAction for more details
#!/usr/bin/python3

import sys
from gi.repository import Gio, Gtk

class PythonApp(Gtk.Application):
    def __init__(self):
        Gtk.Application.__init__(self, application_id="org.example.PythonGnome")
        self.connect("activate", self.on_activate)
        self.connect("startup", self.on_startup)

    def on_startup(self, app):
        self.window = Gtk.ApplicationWindow(application=app)
        hello_world = Gio.SimpleAction(
            name="hello-world", parameter_type=None)
        self.add_action(hello_world)

# Continued on next slide.
# Continued from previous slide.

```python
self.add_accelerator('primary_h', 'app.hello-world', None)
hello_world.connect('activate', self.on_hello_world)

quit = Gio.SimpleAction(
    name='quit', parameter_type=None)
self.add_action(quit)
self.add_accelerator('primary_q', 'app.quit', None)
quit.connect('activate', self.on_quit)

def on_activate(self, app):
    self.window.show_all()

def on_hello_world(self, action=None, param=None):
    print("Hello world!")

def on_quit(self, action=None, param=None):
    self.quit()

# Continued on next slide.
```
# Continued from previous slide.

```python
app = PythonApp()
exit_status = app.run(None)
sys.exit(exit_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
#!/usr/bin/python3

import sys
from gi.repository import Gio, Gtk

class PythonApp(Gtk.Application):
    def __init__(self):
        Gtk.Application.__init__(self, application_id="org.example.PythonGnome")
        self.connect("activate", self.on_activate)
        self.connect("startup", self.on_startup)

    def on_startup(self, self, app):
        self.window = Gtk.ApplicationWindow(application=app)
        hello_world = Gio.SimpleAction(
            name="hello-world", parameter_type=None)
        self.add_action(hello_world)

# Continued on next slide.
Continued from previous slide.

```python
self.add_accelerator("<Primary>h", "app.hello-world", None)
hello_world.connect("activate", self.on_hello_world)

quit = Gio.SimpleAction(name="quit", parameter_type=None)
self.add_action(quit)
self.add_accelerator("<Primary>q", "app.quit", None)
quit.connect("activate", self.on_quit)

appmenu = Gio.Menu.new()
section = Gio.Menu.new()
hello_world_item = Gio.MenuItem.new("Hello world!", "app.hello-world")
quit_item = Gio.MenuItem.new("Quit", "app.quit")
appmenu.append_section(None, section)
section.append_item(hello_world_item)
section.append_item(quit_item)
self.set_app_menu(appmenu)
```

Continued on next slide.
# Continued from previous slide.

def on_activate(self, app):
    self.window.show_all()

def on_hello_world(self, action=None, param=None):
    print("Hello world!")

def on_quit(self, action=None, param=None):
    self.quit()

app = PythonApp()
exiit_status = app.run(None)
sys.exit(exit_status)
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.
# Put these code lines in the right place.

button = Gtk.Button(
    label="Hello world!", action_name="app.hello-world")
self.window.add(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. You should know how to load and save text from a file in Python
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 (learn about this with Fedora at FUDCon)
- Look forward to a future of application sandboxing
[Desktop]
Name=My Python App
Comment=Short description of this application
Keywords=python;editor;
Type=Application
Exec=python–gnome–app
Icon=python–gnome–app
Categories=Gtk;GNOME;Utility;
Example autotools build system (configure.ac)

AC_INIT([Python GNOME App],
         [0.1],
         [amigadave@amigadave.com],
         [python–gnome–app],
         [http://fedorapeople.org/cgi/t/amigadave/public_git/python–gnome–app.git/])

AM_INIT_AUTOMAKE([1.11 foreign])

AC_CONFIG_FILES([Makefile])

AC_OUTPUT
dist_bin_SCRIPTS = python−gnome−app

desktopdir = $(datadir)/applications
dist_desktop_DATA = python−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/
- IRC,
  https://wiki.gnome.org/Community/GettingInTouch/IRC
- https://developer.gnome.org/
- https://python-gtk-3-tutorial.readthedocs.org/en/latest/
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

- popovers, header bars, and lots more!
- See my talk “Bringing your GNOME application up-to-date”