GNOME 3 (3.16)
Design decisions, Main Changes, Future Visions

Tobias Mueller
tobiasmue@gnome.org
Board of Directors

The GNOME Foundation is run by a Board of Directors, which is elected annually by the GNOME community, as the GNOME Membership, to carry out much of the GNOME Foundation's tasks.

The meetings of the Board of Directors are posted publicly on the foundation-list mailing list and on the Minutes wiki page for easier access.

Karen Sandler

Jeff Fortin Tam

Ekaterina Gerasimova

Tobias Mueller

Andrea Veri

Sriram Ramkrishna

Marina Zhurakhinskaya
“GNOME”?
“GNOME 3”?
“GNOME 3.0”?
“GNOME 3.16”?
Elegant design
GNU 3: Ease, comfort and control

GNU 3 is an easy and elegant way to use your computer. It is designed to put you in control and bring freedom to everybody. GNU 3 is developed by the GNOME community, a diverse, international group of contributors that is supported by an independent, non-profit foundation.

Make a donation and become a Friend of GNOME!

Your donation will ensure that GNOME continues to be a free and open source desktop by providing resources to developers, software and education for end users, and promotion for GNOME worldwide.

Get involved!

The GNOME Project is a diverse international community which involves hundreds of contributors, many of whom are volunteers. Anyone can contribute to the GNOME!

Latest news

March 25, 2015  March 18, 2015  February 16, 2015
Activities overview
**GtkApplication** is the base class of a Gtk Application. Its primary purpose is to separate your program from `main()`.

`main()` is an operating system implementation detail that is really uninteresting to applications. The philosophy of GtkApplication is that applications are interested in being told what needs to happen, when it needs to happen, in response to actions from the user. The exact mechanism by which the operating system starts applications is uninteresting.

To this end, GtkApplication exposes a set of signals (or virtual functions) that an application should respond to.

- **startup**: sets up the application when it first starts
- **shutdown**: performs shutdown tasks
- **activate**: shows the default first window of the application (like a new document). This corresponds to the application being launched by the desktop environment.
- **open**: opens files and shows them in a new window. This corresponds to someone trying to open a document (or documents) using the application from the file browser, or similar.

When your application starts, the `startup` signal will be fired. This gives you a chance to perform initialisation tasks that are not directly related to showing a new window. After this, depending on how the application is started, either `activate` or `open` will be called next.

GtkApplication defaults to applications being single-instance. If the user attempts to start a second instance of a single-instance application then GtkApplication will signal the first instance and you will receive additional `activate` or `open` signals. In this case, the second instance will exit immediately, without calling `startup` or `shutdown`.

For this reason, you should do essentially no work at all from `main()`. All startup initialisation should be done in `startup`. This avoids wasting work in the second-instance case where the program just exits immediately.

The application will continue to run for as long as it needs to. This is usually for as long as there are any open windows. You can additionally force the application to stay alive using `g_application_hold()`.

When you receive a `shutdown` signal where you can do any necessary cleanup tasks (such as saving files to disk).
To this end, GtkApplication defines a few hooks for an instance application that wish to react to actions from the user.

- `startup()`: called when the program starts. This is useful for initialisation or activation.
- `shutdown()`: called when the program exits, either because it was closed, or because the operating system is shutting down. If a resource is to be removed at this point, the `shutdown()` function is the appropriate place to do it.
- `activate()`: called when a main window has been activated by the user. This is useful for signals.
- `open()`: called when a file is opened. If you are using XDG File Descriptors, this function is called on the XDG File Descriptors.

When your application contains multiple windows, the activate and open methods are called for each window. If activation is not directly related to the first window being launched, you should not call `activate()` in this case. Function calls in `open()` are only done to the application logic which opens the document (or creates a new one). When the document has been opened, you should return control to the application.

In this case, the `activate()` and the `open()` calls are a request to do some actions on behalf of the user, and there are no other signals that are related to these actions. You should always return to the `activate()` or open signals.

For this reason, you should do essentially no work at all from `main()`. All startup initialisation should be done in `startup`. This avoids wasting work in the second-instance case where the program just exits immediately.

The application will continue to run for as long as it needs to. This is usually for as long as there are any open windows. You can additionally force the application to stay alive using `g_application_hold()`.

On shutdown, you receive a `shutdown` signal where you can do any necessary cleanup tasks (such as saving files to disk).

GtkApplication does not implement `main()` for you. You must do so yourself. Your `main()` function should be as small as possible and do almost nothing except creating your GtkApplication and running it. The “real work” should always be done in response to the signals fired by GtkApplication.
Integrated messaging
**GtkApplication** is the base class for applications written for the GTK+ toolkit. The **main()** function is an operating system implementation detail that is really uninteresting to applications. The philosophy of **GtkApplication** is that applications are interested in being told what needs to happen, when it needs to happen, in response to actions from the user. The exact mechanism by which the operating system starts applications is uninteresting.

To this end, **GtkApplication** exposes a set of signals (or virtual functions) that an application should respond to.

- **startup**: sets up the application when it first starts
- **shutdown**: performs shutdown tasks
- **activate**: shows the default first window of the application (like a new document). This corresponds to the application being launched by the desktop environment.
- **open**: opens files and shows them in a new window. This corresponds to someone trying to open a document (or documents) using the application from the file browser, or similar.

When your application starts, the **startup** signal will be fired. This gives you a chance to perform initialisation tasks that are not directly related to showing a new window. After this, depending on how the application is started, either **activate** or **open** will be called next.

**GtkApplication** defaults to applications being single-instance. If the user attempts to start a second instance of a single-instance application then **GtkApplication** will signal the first instance and you will receive additional **activate** or **open** signals. In this case, the second instance will exit immediately, without calling **startup** or **shutdown**.

For this reason, you should do essentially no work at all from **main()**. All startup initialisation should be done in **startup**. This avoids wasting work in the second-instance case where the program just exits immediately.

The application will continue to run for as long as it needs to. This is usually for as long as there are any open windows. You can additionally force the application to stay alive using **g_application_hold()**.

When you receive a **shutdown** signal you can do any necessary cleanup tasks (such as saving files to disk).
Desktop search
with search providers
$4 \times 4 = 16$

g8443

g8054

CIS224 Software Projects: Software Engineering and Research Methods

Search the Web for $4 \times 4$
And more!

_topic-oriented help
_topic-oriented system settings
...

Many small features to improve your experience!
Under the hood

☕ Fast and powerful platform, usable from several languages: C, C++, Python, JavaScript, Vala, ...

☕ Easy theming with CSS
☕ Symbolic icons
(Fallback Mode)
Try it!
https://download.gnome.org/misc/promo-usb/
Tweaking GNOME 3
http://extensions.gnome.org
GNOME Tweak Tool
GNOME 3 effort bootstrapped in 2008
GNOME 3.16 ≠ GNOME 3
GNOME releases

2.2: 2003-02-06
2.24: 2008-09-24
2.26: 2009-03-18
2.28: 2009-09-23
2.30: 2010-03-31
2.32: 2010-09-29
3.0: 2011-04-06
3.2: 2011-09-28
3.4: 2012-03-28
3.6: 2012-09-26
3.8: 2013-03-27
3.10: 2013-09-25
3.12: 2014-03-26
3.16: 2015-03-25
GNOME 3.16 on March 25th
GNOME 3.16

Approximately 1043 people made about 33,525 changes to GNOME

- New notification system
- Updated shell visuals
- Improved developer experience
- ...
```c
#include <gtk/gtk.h>

static void
activate (GtkApplication *app,
           gpointer user_data)
{
  GtkWidget *window;
  GtkWidget *grid;
  GtkWidget *button;

  /* create a new window, and set its title */
  window = gtk_application_window_new (app);
  gtk_window_set_title (GTK_WINDOW (window), "Window");
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  /* Here we construct the container that is going pack our buttons */
  grid = gtk_grid_new ();

  /* Pack the container in the window */
  gtk_container_add (GTK_CONTAINER (window), grid);

  button = gtk_button_new_with_label ("Button 1");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);

  /* Place the first button in the grid cell (0, 0), and make it fill
   * just 1 cell horizontally and vertically (ie no spanning)
   */
  gtk_grid_attach (GTK_GRID (grid), button, 0, 0, 1, 1);

  button = gtk_button_new_with_label ("Button 2");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);
```
Philosophy behind

GNOME
Accessible & usable by everyone

l10n
i18n
Accessibility
Usability
Freedom
Friends of GNOME donations
$19993.9. $6 to go!
Donate now!
“Filtering out extraneous information is one of the basic functions of consciousness”
— Barry Schwarz
freedom $\neq$ choice
IF YOU FORCE THE USER TO BE A PART OF A SECURITY SYSTEM

YOU'RE GONNA HAVE A BAD TIME
Prompts are dubious
Security prompts are wrong
Interrupting the user to make a permanent security decision is EVIL
This connection is untrusted. Would you like to continue anyway?

The identity provided by the chat server cannot be verified.

The certificate is self-signed.

Certificate Details

- Remember this choice for future connections

Cancel  Continue
The software is not signed by a trusted provider.
Do not update this package unless you are sure it is safe to do so.

Malicious software can damage your computer or cause other harm. Are you sure you want to update this package?

[Buttons: Close, Force install]
Abrt found a new update which fix your problem. Please run:
```
pkcon update --repo-enable=fedora --repo-repo=updates-testing tracker-0.14.1-1.fc17.
```
Do you want to continue with reporting bug?

[ ] No

[ ] Yes
Ellisons Law:
For every keystroke or click required to use a crypto feature the userbase declines by half.
NOT SURE IF

B IS 8
# .caffrc -- vim:ft=perl:
# This file is in perl(1) format - see caff(1) for details.

$CONFIG{'owner'} = 'Username';
#$CONFIG{'email'} = '[user]@[domain]';
#$CONFIG{'reply-to'} = 'foo@bla.org';

# You can get your long keyid from
#   gpg --with-colons --list-key <yourkeyid|name|emailaddress...>
#
# If you have a v4 key, it will simply be the last 16 digits of
# your fingerprint.
#
# Example:
#   $CONFIG{'keyid'} = [ qw{FEDCBA9876543210} ];
#   or, if you have more than one key:
#   $CONFIG{'keyid'} = [ qw{0123456789ABCDEF 89ABCDEF76543210} ];
#   $CONFIG{'keyid'} = [ qw{0123456789abcdef 89abcdef76543210} ];

# Select this/these keys to sign with
#$CONFIG{'local-user'} = [ qw{0123456789abcdef 89abcdef76543210} ];

# Additionally encrypt messages for these kevids
LET'S MAKE THEM USE BASE16, OCAML, AND PERL FOR THEIR CRYPTO
I DON'T ALWAYS TARGET USERS

BUT WHEN I DO, IT'S ME
Key Fingerprint
F289 F7BA 977D F414 3AE9
FDFB F70A 0290 6C30 1813
Step 1: Scan QR Code or type fingerprint and click on 'Download' button.
GNOME is people