GUI Programming with GTK+

Second Part

Florian Pelz

E-Mail: pelzflorian@pelzflorian.de

Today...

Distribution

Miniature IMS

Internationalization

Other features

Last time

We talked about

- how to write a C program,
- how to use C and GTK+ for creating windows and widgets,
- how to incorporate the Glade editor into the workflow and
- developing a small software program.

Issues

However,

- end users don't like having to compile their software before using it and
- compiling with many long terminal commands is annoying even for developers.

Maintainers

- Maintainers fill a vital role concerning the software's infrastructure.
- In a software projekt, maintainers make sure the software
  - is easy to configure, compile, test etc.,
  - runs on all supported systems,
  - can be easily delivered to end users and
  - the remaining infrastructure (Web site, wiki, bug tracker, …) works,
  - good and only good patches get accepted,
  - the change log and news are kept clean,
  - all authors and copyrights get mentioned,
  - …
- Maintainers for an operating system use this infrastructure to distribute the software to end users.

Build Systems

- A *build system* compiles the software automatically.
- Independent of the Integrated Development Environment.
- Necessary for lagge projects.
- Well-known build systems are:
  - GNU Autotools
  - CMake
  - Waf, Scons
  - Ant, Gradle, Maven
  - *Meson*

GNU Autotools

- `autoconf` automatizes
  - finding operating system tools,
  - determining operating system features,
  - compile time configuration (such as compiler selection).
- `automake` automatizes
  - compiliing the software,
  - testing (proving?) the software,
  - installing the software and
  - releasing a version of compilable source code.
- Traditional and common.
- For the actual building of the software, `make` is used.

Meson tools

- `meson` automatizes
  - finding operating system tools,
  - determining operating system features,
  - compile time configuration (such as compiler selection).
- `meson` automatizes
  - compiliing the software,
  - testing (proving?) the software,
  - installing the software and
  - releasing a version of compilable source code.
- Traditional and common.
- For the actual building of the software, `ninja` is used.

Meson

- `meson` is an *easy-to-use* build system.
- Meson is declarative and *not* Turing-complete.
- Many GNOME projects are switching from Autotools to Meson.

- Introduction $\longrightarrow$ Handout.
- More information on `http://mesonbuild.com`

Version Control Systems

- Version control systems like `git` manage multiple source code versions.
- They make it possible
  - for multiple developers to work in parallel on the same code,
  - for a single developer to work in parallel on different parts of the code,
  - for anyone to easily check out an old version and
  - to find the version which introduced a bug.

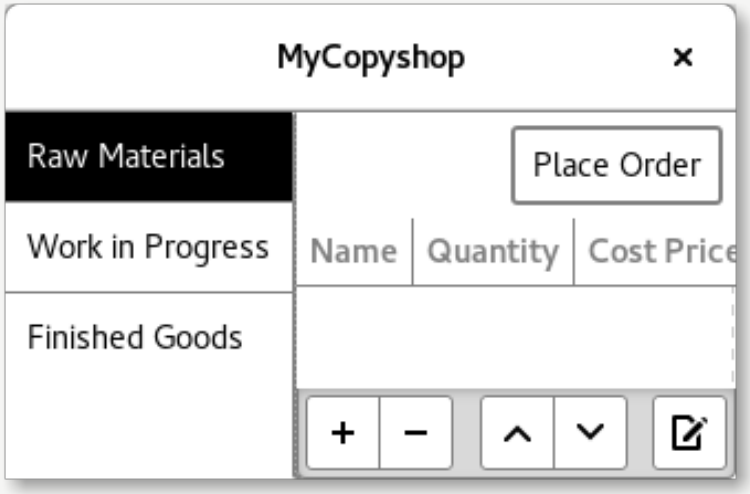
For end users: Package managers

- On GNU systems, software is typically distributed with a package manager.
- Packages mainly consist of
  - either tested, automated scripts for compiling and installing software
  - or the finished files created by such a script, placed at locations appropriate for the operating system.
- Generally packages are created by running a build system.
- Details depend on the package manager.

For end users: Bundles

- For most systems there are also self-containing application bundles (which may need to be installed before use).
- A bundle is a directory containing both the software *and all its dependencies*.
- Metadata and configuration files may be located outside the bundle.
- Such a bundle can be offered e.g. for download on the Web, in an app store, or on physical media.
- Security risk? ($\longrightarrow$ Flatpak)
- Ship together with an offer of making available the "complete and corresponding" GTK+ source code for an appropriately low fee (changed source code must be usable).

Miniature IMS



- For practice we're building a small inventory management system for a fictitious copyshop.
- Since this is a "big project"', we're going to use objects:
  - Inheritance.
  - Modularization.
  - Other languages.

Localization

- Refers to adapting the software to the local language and customs, i.e.
  - translation,
  - representation of date and time,
  - units of measure,
  - decimal marks,
  - ...
- For short: l10n.

Internationalization

- Refers to offering the infrastructure necessary for localization.
- For short: i18n.
- PO files (like “`de.po`”) contain translations for all visible strings.
- In code,

  ```
  label = gtk_label_new ("Hello␣World");
  ```

  must be replaced by

  ```
  label = gtk_label_new (_("Hello␣World"));
  ```

- The `_()` function of GNU Gettext supplies the appropriate translation.
- `_()` can also be interpreted as “marking” the string.
- Similar markings exist to give context to translators such as where in the program this string is being used.
- See handouts.

Accessibility

- That is making software accessible to assistive technology.
- Accessibility means:
  - compatibility with screen readers,
  - compatibility with large text sizes,
  - color choice and contrast,
  - configurable time limit for double clicks,
  - flashing the screen on error,
  - ...
- For short: a11y.
- Mostly automatic in GTK+.
- Still, ATK should be used e.g. for specifying the relationship between widgets.

Unit tests

- Detect program errors through automated testing.
- GLib offers a framework for unit testing.
- Meson can be used to execute tests written with it.
- See for example the source code of GTK+ for an example testsuite.
- reftests compare the user interface pixel by pixel
  $\rightarrow$ used in GTK+.

Static and Dynamic Analysis

- Static: Without running the program.
  - Use multiple compilers.
  - coala for detecting coding style errors.
  - ...
- Dynamic:
  - Valgrind.
  - Profiler.

Profiling

- A profiler measures which parts of the program
  - are executed frequently and
  - how long they take
  - and/or other resource consumption like memory usage.
- This tells you which code to spend time on to optimize performance.

gtk-doc

- Code can be documented inside the source files.
- gtk-doc generates HTML documentation from the source.
- The documentation can be read e.g. using Devhelp.
- But  – GNOME Builder is using reStructuredText/Sphinx, Builder's main author Christian Hergert still is not entirely happy.

Introspection

- Allows for querying information on types and classes:
  - method names,
  - signals,
  - properties,
  - ...
- This enables automatic generation of bindings for other programming languages.

D-Bus

- For Inter-Process Communication (IPC).
- Also used for communicating with the desktop environment:
  - To activate program actions,
  - for notifications,
  - ...

CSS

- CSS allows for changing
  - the style of GTK+ widgets and also
  - configurable keyboard shortcuts

  in GTK+.

More stuff

- GSettings store settings, old program state etc.
- GResource can be used to compile data into the executable file.
- GtkInspector shows and changes GTK+ informationen of running programs.
- `g_autoptr` can be used to automatically free memory when leaving code blocks.
- Mind the Human Interface Guidelines.
- Some things don't need a GUI.

Further reading

- Book on GLib and GTK+:
  `https://people.gnome.org/~swilmet/glib-gtk-dev-platform.pdf`
- For C++ programmers:
  `https://developer.gnome.org/gtkmm-tutorial/stable/`
- Internationalization:
  `https://wiki.gnome.org/TranslationProject/DevGuidelines`

Image sources etc.

- Slide design based on `https://git.gnome.org/browse/presentation-templates/`.
- Construction site caution sign taken from `https://openclipart.org/detail/3850/work` effectively public domain (CC0) by dchandir.