kitlist

Building

kitlist - a program to maintain a simple list of items and assign items to one or more categories.

Required Packages for Desktop Build

Note: Debian 10 (Buster) has dropped the libgconfmm package. By default, the application is now built without GConf. See the 'GConf' section below for more information.

On a Debian 8 (Jessie) or Debian 9 (Stretch) system, the following packages need to be installed to build the application:

- intltool
- autoconf-archive
- libgconfmm-2.6-dev (optional)
- libgtkmm-2.4-dev
- libpqxx-dev (pgsql only)
- libtool
- libxml++2.6-dev
- libglademm-2.4-dev
- libglib2.0-dev (optionally required if needing to re-run `autogen.sh`)
- libyaml-cpp-dev

The application is built and installed with:

```
$ ./configure
$ make
$ sudo make install
```

The build may report warnings in relation to `std::auto_ptr` usage in `libxml++` being deprecated. This is expected.

C++ Compiler Flags

Optionally, to use different C++ compiler flags, set the environment variable CXXFLAGS when running ./configure. E.g.

```
$ CXXFLAGS="-g -O0 -fno-inline" ./configure
```

to include debug messages, define KITLIST_DEBUG

```
$ CXXFLAGS="-g -O0 -fno-inline -DKITLIST_DEBUG" ./configure
```

Exporting the environment variable `G_MESSAGES_DEBUG=ALL` will enable debugging to file. The default location is `/tmp/kitlist.log` which can be amended using either GConf or the `~/.config/kitlist` YAML configuration, depending whether the application was compiled with `--with-gconf`.

Building a Debian Package

On a Debian system, install the `build-essential` package to install the essential packages required to build a Debian package from source. Additionally, install the `devscripts` package to install packages useful for creating Debian packages. As Debian no longer include the GConf package, the Kitlist package should be built without GConf support.

The `./doc` folder in the source distribution should include the following files:

- kitlist.1
- userguide.html
- userguide.pdf

They are removed with `make maintainer-clean`. They can be rebuilt using Pandoc by installing the following packages:

```
- pandoc
- texlive
- texlive-latex-extra
```

then build with:

```
$ ./configure --enable-build-docs
$ make
$ cd doc
$ make docs
```

Build an unsigned package:

```
$ dpkg-buildpackage -us -uc -sa
```

Build GPG signed changes file and source package:

```
$ dpkg-buildpackage -sa
```

If releasing a new version, update the Debian `changelog` with `dch -v version-revision`.

GConf

GConf has been deprecated by the GNOME team. Unless Kitlist is built `--with-gconf`, the application follows the XDG Base Directory Specification storing the configuration settings in `$XDG_CONFIG_HOME/kitlist`, which will typically be `~/.config/kitlst`. The file is YAML formatted.

The following attributes are stored in GConf:

- Printed Page Title
- Most recently used files
- Debug log file name

Running GConf under MAC OS X and Windows is not straight-forward, so the XDG style configuration is used for these targets by default and is probably the better format to use for the future.

To build on Debian 10 (Buster) with `gconf` support, follow the instructions below to build `gconfmm` separately.

1. Install the following Debian package:
   - libgconf2-dev
2. Download the latest version of gconfmm e.g. `gconfmm-2.28.3.tar.xz` and extract and build it under `/usr/local/src`.

   ```
   $ ./configure
   $ make
   $ sudo make install
   $ sudo ldconfig
   ```
3. Build Kitlist:

   ```
   $ ./configure --with-gconf
   $ make
   $ sudo make install
   ```

The `--with-gconf` parameter for `./configure` is not actually required, as `gconf` support is automatically included when the library is found, but by providing the parameter, it ensures an error is generated if the library cannot be found.

Building for Windows

As I no longer have access to a machine running Windows, building on Windows or cross-compiling for Windows is no longer supported.

In the past, the application could be built for Windows platform using MinGW or by cross-compiling on Linux. The instructions remain here in case they should prove useful.

It's not currently documented and not entirely straight-forward, although I have not as yet been able to build the application's alternative language files. However, this fails after the build is otherwise complete, so interrupting the looping make file with Ctrl-c does the trick. In any event you need to:

1. Install the MinGW development environment
2. Install various MinGW packages (TODO:: document which packages) to support the build
3. Install version 2.16 of gtkmm for Windows
4. compile with './configure

Cross Compilation on Linux

The application can be cross-compiled on Linux for a Windows target. These notes are based on instructions for Cross-compiling GTK+ apps for Windows

Setup the tool chain following the instructions on the MinGW Wiki.

The following settings in x86-mingw32-build.sh.conf worked for me:

```
assume GCC_VERSION                  3.4.5-20060117-2
assume BINUTILS_VERSION             2.19.1
assume RUNTIME_VERSION              3.14
assume W32API_VERSION               3.13-mingw32
```

Execute the mingw32 build script with an appropriate target. E.g.:

```
$ sh x86-mingw32-build.sh i686-pc-mingw32
```

Download the gtkmm developer bundle and install it in a new folder using Wine (or Windows), then copy the contents to /opt/mingw/i686-pc-mingw32.

Fix the package config files to have the correct prefix and rename the DLLs.

```
cd /opt/mingw/i686-pc-mingw32
sed -i 's|^prefix=.*$|prefix=/opt/mingw/i686-pc-mingw32|g' lib/pkgconfig/*.pc
cd ./lib
for f in *.lib; do mv $f lib${f%%lib}a; done
```

Finally, build the kitlist application as follows:

```
$ export PATH=/opt/mingw/bin:$PATH PKG_CONFIG_PATH=/opt/mingw/i686-pc-mingw32/lib/pkgcon
$ ./configure --prefix=/opt/mingw/i686-pc-mingw32/ --host=i686-pc-mingw32 --build=i686-p
$ make
$ makensis kitlist.nsi
```

Note: The application does not run under Wine.

Useful Links

- http://www.gtk.org/download-windows.html
- http://live.gnome.org/gtkmm/MSWindows/BuildingGtkmm

Environment Variables

The application can optionally be compiled to use a PostgreSQL database instead of XML documents, using `./configure --disable-xml-dao`. In this case there are a number of environment variables that can be used to specify various connection parameters to the PostgreSQL database. These are listed in the PostgreSQL Documentation. Some of them are mentioned briefly below:

Example Environment Variables

- PGHOST - The database server name
- PGPORT - The port to use
- PGDATABASE - The database name
- PGUSER - The database user name
- PGPASSWORD - The connection password

Internationalisation

1. Translatable strings contained in the program have been written in American English. To create a translation for another language, go to the `po` sub-directory and run the following command to update the default language file `./po/kitlist.pot`:

   ```
   $ cd po
   $ intltool-update --pot
   ```

2. Copy this file to `languagecode.po`, e.g. `fr.po`. This file contains pairs of strings, one in the default language, the other the translated version, initially blank. Also add the language to the list in the `./po/LINGUAS` file, and the `ALL_LINGUAS` entry in `./configur.ac`.

3. To merge code changes into a translated `po` file, e.g. French:

   ```
   $ intltool-update fr
   ```

4. Re-build and re-install the program. To specify the language in a shell, specify the `LANG` environment entry and possibly the `LANGUAGE` environment variable too, e.g.:

   ```
   $ locale -a
   $ LANG=fr_FR.UTF-8 LANGUAGE=fr_FR.UTF-8 kitlist
   ```

   If the locale is not installed, run

   ```
   # dpkg-reconfigure locales
   ```

See https://wiki.debian.org/Locale for more information.

Note: the kitlist program must be installed before the language files are picked up at runtime.

More information in Programming with gtkmm

Using Valgrind

During development and testing, Valgrind can be used to detect memory leaks in the application. The Kitlist source distribution includes the following files:

```
.valrindrc
valgrind-non-kitlist.supp
```

```
valgrind-kitlist.supp
```

`.valgrindrc` contains default values for running `valgrind`.

`valgrind-non-kitlist.supp` contains suppressions for reported leaks that appear unrelated to Kitlist.

`valgrind-kitlist.supp` contains suppressions for reported leaks that are triggered by Kitlist code, but also appear to be outside the control of Kitlist.

Using the defaults in `.valgrindrc`, Valgrind's output is written to `valgrind.log` in the distribution's root directory.

```
$ export CXXFLAGS="-g -O0 -fno-inline"
$ ./configure
$ make
$ valgrind ./src/kitlist
```

After examining the output, to quickly create suppressions based on the log file's contents, `sed` can be used to clean the file. It can then be appended to the suppressions file.

```
$ sed -E -e '/^==[0-9]+.*/d' valgrind.log >> src/valgrind-non-kitlist.supp
```

Documentation

The documentation for the code is maintained using Doxygen. Install the following packages to regenerate the documentation from source:

```
- doxygen
- texlive
- texlive-latex-extra
- texlive-font-utils
```

The documentation can be regenerated from the source code as follows:

```
$ ./configure --enable-build-docs
$ cd doc
$ make docs
```

The generated Doxygen documentation can be viewed under `../doc/doxygen/`.

License

The source code and documentation are licensed under the GPL. See the COPYING and AUTHORS files distributed with the source code for information and contact details.