Xpc66 Developer Guide 0.1.0

Chris Ahlstrom
(ahlstromcj@gmail.com)

April 17, 2024

Xpc66 Logo

Contents

List of Figures

1 Introduction

The *Xpc66* library reworks some of the fundamental code from the *Seq66* project ([2]. This work is in preparation for the version 2 of that project.

Cfg66 contains the following subdirectories of `src` and `include`, each of which holds modules in a namespace of the same name:

- `xpc`. Contains functions for daemonization, message handling, string manipulation, and file manipulation.

In the sections that follow, the basic are described. At some point we will make the effort to add some *Dia* diagrams to make the relationships more clear.

1.1 Naming Conventions

Xpc66 uses some conventions for naming things in this document.

- `$prefix`. The base location for installation of the application and its ancillary data files on *UNIX/Linux/BSD*:
  - `/usr/`
  - `/usr/local/`
- `$winprefix`. The base location for installation of the application and its ancillary data files on *Windows*.
  - `C:/Program Files/`
  - `C:/Program Files (x86)/`
- `$home`. The location of the user's configuration files. Not to be confused with `$HOME`, this is the standard location for configuration files. On a UNIX-style system, it would be `$HOME/.config/appname`. The files would be put into a `po` subdirectory here.
- `$winhome`. This location is different for *Windows*: `C:/Users/user/AppData/Local/PACKAGE`.

1.2 Future Work

- Hammer on this code in *Windows*.

2 Xpc Namespace

This section provides a useful walkthrough of the `xpc` namespace of the *xpc66* library. In addition, a `C`-only module is provided.

Here are the classes (or modules) in this namespace:

- `automutex`
- `condition`
- `daemonize`
- `recmutex`
- `ring_buffer`
- `shellexecute`
- `timing`
- `utilfunctions`

2.1 xpc::automutex

`xpc::automutex` provides a recursive mutex that locks automatically when created, and unlocks when destroyed. This has a couple of benefits. First, it is threadsafe in the face of exception handling. Secondly, locking can be done with just one line of code.

It could potentially be replaced by `std::lock_guard<std::recursive_mutex>`. One reason we rolled our own was some difficulty experienced using the standard mutex in the *Seq66* ([2]) application.

The constructor takes a reference to an `xpc::recmutex` (see below), stores it, and locks it. The destructor simply unlocks it.

2.2 xpc::condition

`xpc::condition` provides an internal recursive mutex and a private implementation of the `wait()` and `signal()` functions. The implementation uses a `pthread_cond_t` condition variable and `xpc::recmutex` to implement these functions.

Also provided is the more useful and simpler `xpc::synchronizer` *abstract base class* which uses an `std::mutex` and an `std::condition_variable` to implement the `wait()` and `signal()` functions. It requires the caller to derive a class which implements the *virtual* function `predicate()` that decides when synchronization has occurred.

For a good example of `xpc::synchronizer`, see the `seq66::performer::synch` class defined in the `performer` module of the *Seq66* project.

2.3 xpc::daemonize

This module implements demonization code as described in *The Linux Programming Interface* ([1]). It provides many options as expressed by the `daemonize_flags` enumeration:

- Don't chdir() to the file root directory '/'.
- Don't close all open files.
- No stdin etc. sent to /dev/null.
- Don't call umask(0).
- Don't call fork() a second time.
- Don't change current directory.
- Do not open a system log file.

The most important functions are `daemonize()` and `undaemonize()`. For the usage of these functions, see the main module `seq66rtcli` in the *Seq66* project.

Also provided in this module are functions for getting process information, rerouting standard I/O, and flagging session saving, restart, and closing.

Note that this is a `C++`-only module using `std::string` to pass and store information.

2.4 xpc::recmutex

This recursive mutex is implemented using `pthread_mutex_t` due to difficulties we had with `C++11`'s `std::mutex` in the *Seq66* project.

Read the module's comments for more information on the ifs, ands, buts, or maybes..

2.5 xpc::ring_buffer

This template class defines a flexible ring-buffer. It support reading and writing, skipping, and the `front()` and `back()` functions.

The `ring_buffer.cpp` file contains an explanation of the implementation and some code to test the ring-buffer.

2.6 xpc::shellexecute

This module provides free functions in the `xpc` namespace for spawning applications and opening PDFs and URLs. These functions provide support for *Linux/UNIX* and `Windows`.

```
command_line (const std::string & cmdline)
open_document (const std::string & name)
open_pdf (const std::string & pdfspec)
open_url (const std::string & pdfspec)
open_local_url (const std::string & pdfspec)
```

2.7 xpc::timing

This module provides free functions in the `xpc` namespace for getting the system time and for sleeping. These functions provide support for *Linux/UNIX* and `Windows`.

```
std_sleep_us ()
microsleep (int us)
millisleep (int ms)
thread_yield ()
microtime ()
millitime ()
set_thread_priority (std::thread & t, int p)
set_timer_services (bool on)
```

More explanation can be found in `timing.cpp`.

2.8 xpc::utilfunctions

In order to keep the *Xpc66 library* independent of the *Cfg66 library*, this module provides cut-down versions of the message functions of the latter. It also uses some code to work with directories, getting the date/time, and widening ASCII strings.

3 Xfg66 Tests

This section provides a useful walkthrough of the testing of the *xpc66* library. They illustrate the various ways in which the *Xfg66* library can be used by a developer.

The tests so far are these executables:

- `xpc_tests`

These tests are supported by data structures define in the following header files:

- textttodo.hpp

These header files will be discussed as needed in the following sections.

3.1 Xfg66 Test

TO DO.

Obviously, we still have a lot of work to do with these tests.

4 Summary

Contact: If you have ideas about *Xpc66* or a bug report, please email us (at `mailto:ahlstromcj@gmail.com`).

5 References

The *Cfg66* reference list.

References

[1] Michael Kerrisk. *The Linux Programming Interface* `https://man7.org/tlpi/` 2010.

[2] Chris Ahlstrom. *A reboot of the Seq24 project as "Seq66".* `https://github.com/ahlstromcj/seq66/`. 2015-2024.