xy Documentation

Release 2012

Nick Bargnesi

April 27, 2013

CONTENTS

This is the documentation for XY 2012, last updated February 01, 2013.

CHAPTER

ONE

INTRODUCTION

XY is an environment for GNU/Linux and a playground for innovative computer interfacing that aims to integrate the latest technology and hardware.

Website: http://bb.den-4.com/xy

Bugs: http://bb.den-4.com/xy/issues

Documentation: http://xy.readthedocs.org

CHAPTER

TWO

CONTENTS

2.1 Overview

2.1.1 Directories

docs

The documentation for xy resides in the *docs* directory.

XY's documentation is managed by Sphinx; the latest version is always available online.

examples

Various examples for interacting and integrating with xy live in the *examples* directory.

lib

The core of xy is housed in the *lib* directory. The real parts of xy live here – from utilities and library interefaces to modules like xy's broadcasting engine.

resources

The *resources* directory contains xy's ancillary files. Resources for build automation and testing, images, and runtime resources are all held here.

src

The *src* directory contains xy's *main* function - the application's entry point. The main executable is built independently of xy's libraries and tests to allow for things like parallel builds and a robust level of testing.

tests

The *tests* directory contains binaries that test pieces of xy. The same tests are used to exercise various components and identify potential memory leaks or performance problems.

XY supports testing via the Check unit testing framework.

2.1.2 Requirements

TODO - requirements currently in flux (njb)

2.1.3 Building

The xy source tree is managed by Autotools. It follows a standard configure-and-build approach.

```
# ./configure --quiet
xy build configuration
----------------------
version         : 2012
cflags          : -pipe -std=c99 -Wall -Wunused -Wextra -Wno-unused-parameter -O0 -g3
tests           : yes
optimizations   : no
```

The default build configuration targets developers. To build xy for production use:

```
# ./configure --enable-optimizations --disable-tests --quiet
xy build configuration
----------------------
version         : 2012
cflags          : -pipe -std=c99 -Wall -Wunused -Wextra -Wno-unused-parameter -O3
tests           : no
optimizations   : yes
```

2.1.4 Running

XY can be launched using *ck-launch-session* and a `Xclients` or `xsession` file in your home directory. Your login manager can also be configured to start xy for you. Refer to your distribution's documentation for more information.

Under Valgrind

You can run xy under Valgrind, to help isolate memory leaks or troubleshoot performance problems. The overhead of running xy with Valgrind will make the interface cumbersome making xy unsuitable for regular use.

```
valgrind --tool=memcheck --leak-check=full --log-file=$HOME/valgrind.xy.log xy
```

2.1.5 Documenting

Documentation can be generated by using the *html* make target.

```
# make html
```

Use of this target requires Sphinx. The HTML-based documentation will be located in `docs/build/html`.

2.2 API Reference

2.2.1 configuration.h

The configuration module reponsible for managing the configuration of xy. Any change to xy's runtime configuration (RC) file will cause the configuration to be reloaded (see `config_reinit()`).

CONFIG
The type made available after a call to `config_init()`. It contains all the necessary configuration options xy uses while running.

CONFIG * **config_init**()
Initializes xy's configuration. Returns the configuration or NULL on failure. This function makes `CONFIG` available for use.

void **config_terminate**()
The configuration module's shutdown hook.

void **write_default_config**()
Writes the default xy configuration to the xy runtime configuration path.

void **config_reinit**()
Reinitializes the configuration from the xy runtime configuration path.

2.2.2 util.h

A collection of utility functions meant to stand alone from other xy modules.

char * **rc_path**()
Returns the path of xy's runtime configuration. The returned string should be freed by the caller.

bool **streq**(const char *, const char *)
Returns `true` if the strings are equal or null, `false` otherwise.

char * **trim**(char *)
Removes leading and trailing whitespace from the string and returns a pointer starting from the first non-whitespace character. The returned string is contained in the same memory as the argument.

Example:

```
char *conststr = " test string ";
char *dup = strdup(conststr);
char *trimmed = trim(dup); // trimmed: "test string"
free(dup);
```

void **dump_stack**(uint *num_frames*)
Dumps *num_frames* stack frames to stderr.

void **parse_command**(char **cmd*, char ***argv*)
Tokenizes *cmd* by spaces into tokens placed in *argv*.

Example:

```
char *command = "some string here";
char *dup = strdup(command);
char *argv[3];
parse_command(dup, argv);
// argv[0]: "some"
// argv[1]: "string"
// argv[2]: "here"
free(dup);
```

void **exec**(const char * *cmd*)
Executes the supplied command by calling *execvp*. The command will be parsed before the call is made (see `parse_command()`).

2.2.3 inotify.h

The inotify module. XY uses the inotify module to react to changes occurring on the filesystem.

types

in_fd
The inotify file descriptor made available after a call to `xy_inotify_init()`. This file descriptor is suitable for system calls like *select* and *epoll*.

functions

void **xy_inotify_init**()
Initializes xy's inotify module. This function makes `in_fd` available for use, which may be -1 on failure.

void **xy_inotify_reinit**()
Reinitializes the inotify module. This function makes a new `in_fd` available for use.

void **xy_inotify_terminate**()
The inotify module's shutdown hook.

void **xy_inotify_read**()
Drains the inotify event queue of all events.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*